THE SOLUTION OF A MILK-TRUCK ROUTING PROBLEM VIA TRAVELING SALESMAN ANALYSIS: AND THE DEVELOPMENT OF AN ALTERNATIVE APPROACH

by

WALTER LYNN TURNER

B. S., Kansas State University, 1974

A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF BUSINESS ADMINISTRATION

College of Business Administration

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1976

Approved by:

John R. Graham

Major Professor

TABLE OF CONTENTS

Page

LIST OF FIGURES AND TABLES iv

ACKNOWLEDGEMENTS v

I. INTRODUCTION 1

A. Problem Description 1
B. Cost-Link Matrix Development 2

II. LITERATURE SEARCH 6

A. Tour to Tour Improvement 6
B. Tour Building 7
Dynamic Programming 8
Branch and Bound 8
Heuristics 9
C. Subtour Elimination 10
Assignment Problem Approach 11
Integer Programming 11

III. METHODS 15

A. Integer Program Model 15
B. Problem Solution Attempt 20
C. An Alternative Approach 22
D. CITDIS Development 23

IV. RESULTS 32

V. CONCLUSIONS 36

A. CITDIS Improvements 37
B. TS Analysis Benefits 37

VI. LITERATURE CITED 39

Page

VII. APPENDIX 42

Appendix A. PAUL I 43
Appendix B. CITDIS 54

ABSTRACT

LIST OF TABLES AND FIGURES

Page

Table 1 Cost-link Matrix for the Eleven-city Milk Route 3

Table 2 Comparison of Original and Shortest Routes 33

Table 3 Cost-link Matrix for the Twelve-city Milk Route 35

Figure 1 Subtour Example 17

Figure 2 RIP 30C City Size Versus Computation Time 21

Figure 3 Routes in Order of Generation for a Six-city Problem 24

Figure 4 CITDIS City Size Versus Computation Time 31

ACKNOWLEDGEMENTS

The author wishes to express his grateful appreciation to Ed Reutzel, College of Business Administration, for his valuable advice during this investigation and for his guidance in the preparation of this thesis.

Acknowledgement is due Dr. John Graham, Dr. Jim Gentry, Dr. Richard Vaden, and Dr. Tom Brown, College of Business Administration, for their critical review of this manuscript.

Sincere appreciation is expressed to Earl Glynn and Paul Mensch, consultants at the Kansas State University Computing Center, for their help in using RIP 30C and the writing of a program (Paul I) to prepare RIP 30C's input data.

I. INTRODUCTION

The traveling salesman problem (how to visit M cities, passing through each city only once and returning to the point of origin in the shortest distance possible) was first proposed by Hassler and Whitney (Flood, [16]) at a Princeton seminar in 1934. However, until recently, its resolution has been chiefly an academic exercise due to the time involved in solving traveling salesman (TS) problems and the lack of economic pressure to cut costs and hence travel distance. Lately, the problem has become more relevant as energy shortages and economic factors have caused fuel, cost, and time savings to be of more importance. Also, the advent of high speed computers has made the solution of TS problems more practical. In view of the increasing relevance of the traveling salesman problem, this paper is oriented toward investigating the business and technical benefits/costs inherent in the solution of a real-world TS problem and the development of a specialized technique to solve it.

Problem Description

As any milk delivery system would be a TS problem, the routing of a milk delivery truck for the Foremost Dairy of Topeka, Kansas, was chosen for consideration. The route being driven was to be compared to the shortest route. If the route being driven was not the shortest, then the good and bad points of switching to the shortest route were considered.

The route used by the truck under consideration consisted of twelve stops. The truck left the dairy at 1) Topeka, went to 2) Auburn, on to 3) Carbondale, then to 4) Overbrook, on to 5) Michigan Valley and then to 6) Pomona. Leaving Pomona, the truck then stopped at 7) Green Acres, then 8) Vassar-Hedgewood Acres, then to 9) Osage City, then 10) Burlingame, then 11) Scranton, and finally back to 1) Topeka--a total of 116.5 miles. In terms of city numbers used in solving the problem, the route could be stated as 1-2-3-4-5-6-7-8-9-10-11-1.

Consideration of alternative routes, in hopes of finding one shorter, consisted of examining county maps, selecting routes linking the various cities, and then recording the mileage (cost) of taking a given road (link) in going between two cities. The choice of possible city links was constrained in that only paved roads were allowed [except for the gravel road then being used between 5) Michigan Valley and 6) Pomona] as milk spillage problems were known to result from the use of non-paved roads. Certain obviously bad links, such as going from a city on one side of the existing route around all the other cities to get to a distant one, were eliminated by visual observation.

Cost-Link Matrix Development

A convenient way to depict the possible routings and associated costs is via a cost-link matrix. This was done for the TS problem at hand and is shown in Table 1. Entries in the cost-link matrix represent the cost, expressed in miles, associated with completing a link from one

THIS BOOK CONTAINS NUMEROUS PAGES WITH DIAGRAMS THAT ARE CROOKED COMPARED TO THE REST OF THE INFORMATION ON THE PAGE.

THIS IS AS RECEIVED FROM CUSTOMER.

Table 1

Cost-Link Matrix for the Eleven-City Milk Route

(Miles between Cities)

From City i \ To City j	1	2	3	4	5	6	7	8	9	10	11
1	∞	17.5	16.5	∞	∞	∞	∞	∞	∞	26.0	∞
2	17.5	∞	13.5	∞	∞	∞	∞	∞	∞	11.0	∞
3	16.5	13.5	∞	9.5	∞	30.5	∞	15.5	20.0	12.5	5.5
4	∞	∞	9.5	∞	8.0	17.0	∞	15.5	24.5	∞	10.0
5	∞	∞	∞	8.0	∞	9.0	4.5	7.5	∞	∞	∞
6	∞	∞	30.5	17.0	9.0	∞	7.5	10.5	25.0	∞	30.0
7	∞	∞	∞	∞	4.5	7.5	∞	3.0	∞	∞	∞
8	∞	∞	15.5	15.5	7.5	10.5	3.0	∞	10.5	∞	16.0
9	∞	∞	20.0	24.5	∞	25.0	∞	10.5	∞	9.0	20.5
10	26.0	11.0	12.5	∞	∞	∞	∞	∞	9.0	∞	7.0
11	∞	∞	5.5	10.0	∞	30.0	∞	16.0	20.5	7.0	∞

city to another. Each row represents departure from city i and each column represents arrival at city j, with the value located at the corresponding position in the matrix representing the cost (miles) associated with traveling between the two cities. For this problem, the distance *from* city i to city j equals the distance from city j to city i. This would not necessarily always be the case due to limitations such as bridge clearances, one-way streets, etc.

The diagonal of the matrix (as one would not go from a city to itself) and any other position in the matrix which has not been assigned a finite numeric value implies that there is no direct link from the city represented by that row to the city represented by that column *without* passing through another city in the problem. Thus, the cost assigned to these links is infinity to preclude them from ever occurring. However, in the case of cities such as Pomona, where backtracking (using the same road twice) occurs, the route may need to pass through a previously visited city simply because that city lies on the shortest route from one city to another city which has not previously been visited. This is accomplished by selecting the next city (C) on the road past the city that must be backtracked through (B). The link representing travel from the first city (A) through (B) and on to (C) is then allowed by placing the appropriate distance in the cost link matrix at the intersection of row *A and column* C. Even with these backtrack lines included, the size of this problem is considerably smaller than the hypothetical case of all links between all cities being valid.

Many solution techniques exist to solve this particular problem and indeed the general traveling salesman problem. The various approaches will be reviewed in the next section.

II. LITERATURE SEARCH

The proposed approaches to solving the traveling salesman problem are many, and they have had varied degrees of success. According to Bellmore and Nemhauser [6], the approaches fall into three basic categories: 1) Tour to tour improvement, all of which are approximations of the exact answer; 2) Tour building; and 3) Subtour elimination, both of which contain exact algorithms and approximate heuristics. These categorical divisions will be used herein to describe the various approaches. When solution times are given for a certain size problem, an attempt is made to cite the computer used when that information was available. Even so, direct comparison is difficult because times on the same type of computer will vary with the environment, the model, the number of jobs in the system when a problem was run, the amount of fast core, and the total amount of main core storage. However, for rough comparisons, it can safely be said that the IBM 7000 series is at least fifteen times slower than the IBM 360 series. The 360 series, in turn, is at least five times slower than the 370 series.

Tour to Tour Improvement

Algorithms of this type use tours formed combinatorially and try to improve on a given tour length by interchanging links in completed tours or by analyzing a given partial tour to form the best completion of it. All algorithms written to date, using a tour to tour improvement approach, have yielded approximate answers.

One method (discussed by Richmond [31]) which amounts to computer simulation, is to have a computer consider some number of possible random solutions picked via a random number generator. Then, the mean and standard deviation of each solution's total route distance is found and the probability that another lower solution exists is determined along with an estimate of how much lower the solution would be. This next lower solution is then found and so on until the estimated gain represented by an estimated lower solution is exceeded by the estimated cost of finding that lower solution.

Another approach, originally developed by Croes [11], is a systematic way of judging which links in an initial tour, should be interchanged with others. It is based on the assumption that links occurring in many good tours should occur in the optimal tour. This method shows promise in some of the improved versions. An improvement by Lin [26] has solved a forty-eight-city problem in sixty-three seconds although the author is unable to prove the solution to be optimal. A further improvement by Christofides and Eilon [8] consists of subdividing the problem into sub-areas and then replacing links in the initial tour for each sub-area, finally reconnecting them to form a completed tour. Christofides and Eilon have solved problems up to 500 cities in size in 152 seconds, yielding approximate solutions.

Tour Building

Approaches of this type start with a portion of a tour and add links, based on various selection rules, to achieve a completed tour. In certain algorithms, if the completed tour is not optimal or within a certain range of optimality, tour generation starts again.

Dynamic Programming

Tour building algorithms of this sort work by eliminating permutations by comparing them and picking the shortest via solving recursive equations. The biggest drawback is that the various partial solutions generated in the process must all be stored--using over 32K bytes for a thirteen-city problem. Conway, Maxwell and Miller [10] report that the storage requirements more than double as the size increases. Lin [26] reports using dynamic programming to solve a thirteen-city problem in 1.75 seconds. Bellmore and Nemhauser [6] report the solution of a thirteen-city problem in .28 seconds. Held and Karp [22] have also used it in their spanning-tree algorithm to solve a twenty-five-city problem; however, they recommend coupling dynamic programming with some sort of branch and bound system to increase efficiency. As assignment problems would be solved with dynamic programming, portions of the tree (see next section) could be eliminated.

Branch and Bound

In algorithms of this type, the possible cities available to visit, given M cities have been visited, form a tree with the branches representing choices of visiting a certain city from another. Algorithms of this sort trace through the tree while calculating feasible bounds (limits) on the optimal route. The tracing of a given branch terminates when the distance along it exceeds the lowest upper bound thus far calculated. One such procedure by Little [27] has solved (on an IBM 7090) one hundred 30 city TS problems in a mean time of 58.5 seconds and five 40 city problems in a mean time of 8.37 minutes. Little notes a time increase of roughly ten times for every ten city increase in problem size.

Another branch and bound tour building technique by Svestha and Huckfeldt [33], although designed for M-salesman TS problems, has been used to solve a 1-salesman, sixty-city problem in eighty seconds.

Heuristics

In addition to the above exact approaches, there are many heuristic tour building methods. Although these do not always yield the optimal solution (or the solution can not be proven optimal), any answer better than an existing one has potential benefits.

One such approach, by Barachet [4], considers city links that appear to be obviously good. These are taken in groups 1,2,3...n at a time until a complete tour is picked. By applying geometric rules (such as the finding that the optimal tour does not cross itself) along the way and always selecting the next link such that the group distance is minimized, a near optimal solution is hopefully found.

Another geometric approach, by Nicholson [29], solves TS problems by forcing the tour to go through the cities such that it forms their boundary with no tours crossing. The cities are picked on the basis of being the next closest city that meets the above criterion. Tours within one percent of optimality have been found with this method, although accuracy will tend to vary with the spatial distribution of the cities.

A heuristic by Conway, Maxwell, and Miller [10] also use the closest unvisited city as a rule, but without the geometric rules. This technique has been applied to twenty-city problems, yielding solutions that average within 26% of optimality. Rotation of the starting city among all cities increases the computational time by a factor of n, but increases accuracy in a twenty-city problem to within eighteen percent of optimality.

Yet another heuristic by Karg and Thompson [25] works by picking two cities, then inserting another between them so that the 3-link tour length is minimal. Then a fourth city is picked and so on until every city is in the tour. As a tour is completed, the probability of a shorter tour existing is calculated and, if high enough, a different pair of starting cities is selected and the procedure repeated until the answer is as close to optimal as desired. Utilizing this method, a fifty-seven-city problem was solved to within thirty miles of the optimal tour in seventeen minutes, on a Bendix G-20 computer.

Finally, a heuristic approach by Webb [35] using five sequential algorithms has solved 2500 city problems to within twenty-five percent of the assignment solution (lower bound) in 140 seconds.

Subtour Elimination

The published, accessible algorithms for subtour elimination consist of approaches based on solving assignment problems and modifying solutions based on subtour constraints, and techniques based on integer programming.

Those involving integer programming use subtour constraints to reject tour links that would cause subtours to be formed, whereas the assignment problem based algorithms incorporate subtour constraints to reject completed solutions containing subtours. (For a more complete explanation of subtours, see the Methods section.)

Assignment Problem Approach

A published example of an algorithm based on solving successive assignment problems while considering subtour constraints is the one by Bellmore and Malone [5]. They break the cities into subsets and then proceed to solve assignment problems involving the cities in each subset. The best solution to a subproblem is kept (one with no subtours) along with the subproblems with subtour solutions. The subproblems with subtours are further divided and assignment problems solved for these subsubproblems. In this manner, at least one feasible solution is eliminated with every solution of subproblems, so the problem must eventually converge. Bellmore and Malone report solving exactly a 182 city random-asymmetric TS problem in sixty minutes on an IBM 360-65 computer. Symmetric problems reportedly take longer. Also, they predict that if a problem had no feasible solution that their algorithm would take an excessive time to find this out.

Integer Programming

There are two basic approaches to integer programming. One I will deem the constraint addition approach, and the other using branch and bound methods. Dantzig, Fulkerson, and Johnson [12,13] are credited with the solution of one of the earliest, and the largest to date, integer programming problem. They solved exactly a forty-two-city problem using linear programming (in seventy hours by hand) by stopping every few iterations and considering subtour constraints to eliminate any subtours or fractional answers and then continuing to the next iteration. This was repeated until the only feasible tour remaining was also the shortest. Martin, as reported by Bellmore and Nemhauser [6], has solved the same problem in under five minutes on an IBM 7094. He started with forty-two subtour constraints and then added more as needed.

Balas [3] can be credited with first applying branch and bound or "implicit enumeration" procedures to integer programming. His method aligns the 2^n possible 0 - 1 variable values, using binary expansion for values that could take on larger values, in a tree and then searches it, implicitly eliminating certain branches. This is done by assigning those variables whose value is known either as 0 or as 1 and then trying the other variables both as 0 and as 1 in separate problems--which constitutes the branching. The bounding of the variables is done by taking the sum of the costs of those variables known to be 1 and those being tested as 1. Once an upper bound is set, many combinations exceeding the bound can be implicitly eliminated. This continues down the tree until a terminal point is reached, at which time an answer to the problem or the fact that it has no answer can be determined. Balas notes that the fewer the number of feasible answers a problem has, the longer the tree search would take, as less could be implicitly searched by rejecting feasible solutions and their associated infeasible ones. The largest problem he reports solving, by hand, is one of fifteen variables and twelve constraints (non TS).

Freeman [17] reports using a modification of Balas's algorithm to solve problems of from fourteen variables and four constraints to twenty variables and six constraints in times ranging from two seconds to seven minutes on an IBM 7090.

Another modification of Balas's algorithm was tested in 1967, on an IBM 7094, by Fleischman [15]. He used it to solve various types of integer programming problems, including two TS problems. The TS problems, one a six-city problem (thirty-five variables, thirty-one constraints) and the other a seven-city problem (forty-eight variables, forty-three constraints) were solved in .86 minutes and 5.75 minutes (13,790 iterations) respectively.

Geoffrion [19], in 1968, developed another algorithm based on Balas's methods, using an imbedded linear program to speed up the branch and bound searches. After solving various sized problems with it, he noted that computation time did not increase exponentially with problem size, as had been the case with other approaches, but that the increase was that of a low order polynomial. He reports solving a problem with seventy-four variables and thirty-seven constraints in slightly over ten minutes on an IBM 7044.

In view of the possible potential of this last approach to the TS problem, and the ready availability of Geoffrion's algorithm, RIP 30C [18], it was decided to attempt to solve the milk route problem using this algorithm. (For a better explanation of how RIP 30C works, see [30].) It was hoped that reasonable times could be achieved on the eleven-city milk route problem (ninety-eight variables, ninety-six constraints) through the use of an IBM 370-158, which is many times faster and has a larger main core storage than the older IBM 7044 used by Geoffrion.

III. METHODS

Before RIP 30C could be applied to the milk route TS problem, the problem had to be expressed as a series of equations amenable to binary integer programming. The basic model is similar to that of an assignment problem. The model can be expressed as follows [28]:

$$(1)\ \min \sum_{i=1}^{11} \sum_{j=1}^{11} C_{ij}X_{ij} \text{ where } C_{ii}= \quad \text{for } i=1,2,\ldots 11$$

Subject to:

$$(2)\ \sum_{i=1}^{11} X_{ij} = 1 \qquad \text{for } j = 1,2,\ldots 11 \text{ (arrival)}$$

$$(3)\ \sum_{j=1}^{11} X_{ij} = 1 \qquad \text{for } i = 1,2,\ldots 11 \text{ (departure)}$$

$$X_{ij} = \begin{cases} 1, & \text{if city } i \text{ is visited from city } j \\ 0, & \text{otherwise} \end{cases}$$

Integer Program Model Explanation

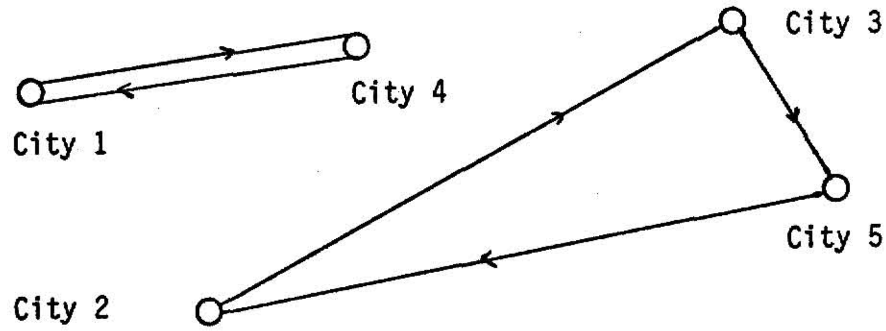
This model is a shorthand form for the following: (1) the total of the costs C_{ij} (in this case miles) for each link X_{ij} completed (with $X_{ij}=1$ representing travel from city i to city j) should be the minimal amount possible subject to (2) and (3) above; (2) the sum of each column in the cost-link matrix should be equal to 1; and (3) the sum of each row in the cost-link matrix should be equal to 1. By limiting the sum of the rows and columns to 1, (2) and (3) insure that each city is entered and departed only once. For example, if the link from city 1 to city 2 is picked, the variable representing that link, X_{12}, taken on the value 1, which represents a decision to use that link. The 1, in turn, keeps any other link representing travel from city 1 to a city other than city 2 from being picked because, if both were picked, two variables in the same row would equal 1 and cause the sum of the variables for the row to exceed one. Conversely, this would work for columns and prevent the departure from a given city to more than one city.

Sometimes the solution to a TS problem and the corresponding assignment are the same. However, as the number of cities in a problem increases, the probability of the assignment problem solution being a complete tour decreases, [5]. Because the assignment solution often contains subtours (a route containing less than all the cities in a problem), they must be kept from occurring via constraining equations. For example, for a five-city problem, the routes 1-4-1 and 2-3-5-2 would meet the requirements of (2) and (3) above but would not form a complete tour through all the cities as is required in a TS problem solution. (See Figure 1.)

One way to eliminate the subtours is by introducing a series of equations that, in essence, block each particular subtour. For example, a subtour from city 1 to city 2 and back to city 1 could be blocked by the equation $X_{12} + X_{21} \leq 1$. In order for this equation not to be violated, both X_{12} and X_{21} can not have the value 1. Thus, if the link from city 1 to city 2 is used, the link from city 2 back to city 1 can not be used, and vice versa. This system of blocking subtours would result in $2^{n-1}-1$ [6] total problem constraints, which is a sizable number for even a fairly

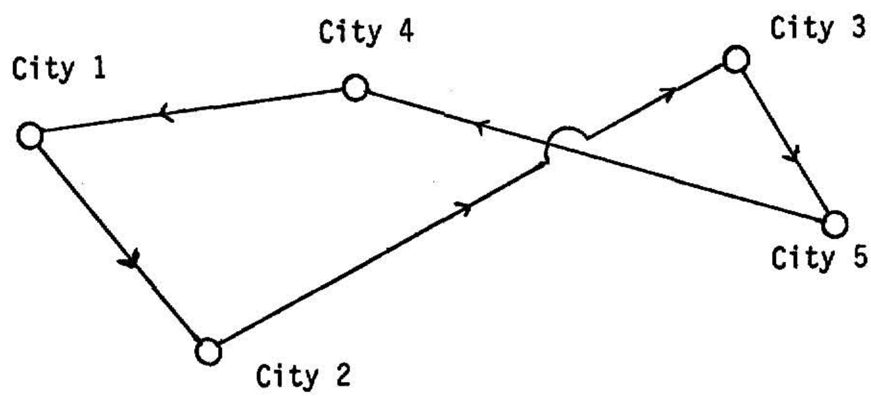
Figure 1

Subtour Example



Subtours: 1-4-1 and 2-3-5-2

Completed Tour



Tour: 1-2-3-5-4-1

small problem.

A shorter way of representing these subtour constraints, as shown by Wagner [34], is to introduce a series of equations of the following form:

$$(4) \quad u_i - u_j + nX_{ij} \leq n - 1 \quad \text{for } i = 2,3,\ldots n \quad j = 2,3,\ldots n$$

where u_i and u_j are non-negative variables,

and n = the number of cities in the problem.

For an example of how this set of equations works to constrain subtours, consider the subtour: City 4 to city 5 to city 6 to city 4 in the example. This would be represented by the variables $X_{45} = X_{56} = X_{64} = 1$. The associated subtour constraints are:

$$u_4 - u_5 + 11X_{45} \leq 10$$
$$u_5 - u_6 + 11X_{56} \leq 10$$
$$u_6 - u_5 + 11X_{64} \leq 10.$$

Adding these inequalities, one obtains the composite constraint: $(u_4 - u_5 + u_5 - u_6 + u_6 - u_4) + 11(X_{45} + X_{56} + X_{64}) \leq 30$. This further reduces to: $11(X_{45} + X_{56} + X_{64}) \leq 30$. If $X_{45} = X_{56} = X_{64} = 1$ occurred, the left hand side of the constraint would equal 33 and the constraint would be violated. Thus, the $n - 1$ equations represented by (4) above combine to eliminate the subtours.

The complete formulation of model equations (2), (3), and (4) requires $n^2 - n + 2$ linear constraints [34]. For an eleven-city problem with all links between cities available for consideration, this computes to 112 constraints. However, the nature of RIP 30C required that an additional twenty-two constraints be formulated. RIP 30C requires all constraints to be of the form $g(x) \geq 0$ or $g(x) \geq 0$ and hence the equalities in (2) and (3) above had to be replaced by two inequalities, thus replacing the eleven column and eleven row constraints with twenty-two column and twenty-two row constraints. Total constraints therefore, for an eleven-city problem with all links usable, would number 134.

The number of variables necessary to represent a TS problem can be computed by the formula $n^2 + n - 1$, [34]. This totals to 131 for an eleven-city problem with all links between cities being considered for traverse. Of this, eleven variables can be eliminated since they represent a trip from a city to itself (X_{ij} where $i = j$). For RIP 30C, the u_i and u_j variables in the subtour constraints are subjected to a binary expansion so that only 0 - 1 variables exist. While the u_i need not be integer, no harm is done if they are restricted as such. Specifically, since the bounds on the variable u are

$$0 \leq u \leq n, \text{ where } 2^{k-1} \leq n \leq 2^k;$$

then each feasible value of u can be expressed uniquely as

$$u = \sum_{i=0}^{k} 2^i y_i ,$$

where the y_i variables are restricted to be either zero or 1, [23]. For an eleven-city problem with all links usable, this would amount to four y_i variables in each of the ten subtour constraints (thus forty in all) for a total of 150 0 - 1 variables. In the particular eleven-city problem at hand, the missing links between cities allowed the removal of constraints and variables that would have been associated with those links.

Thus, the eleven-city milk route problem can be expressed with ninety-eight variables in ninety-six constraints.

Problem Solution Attempt

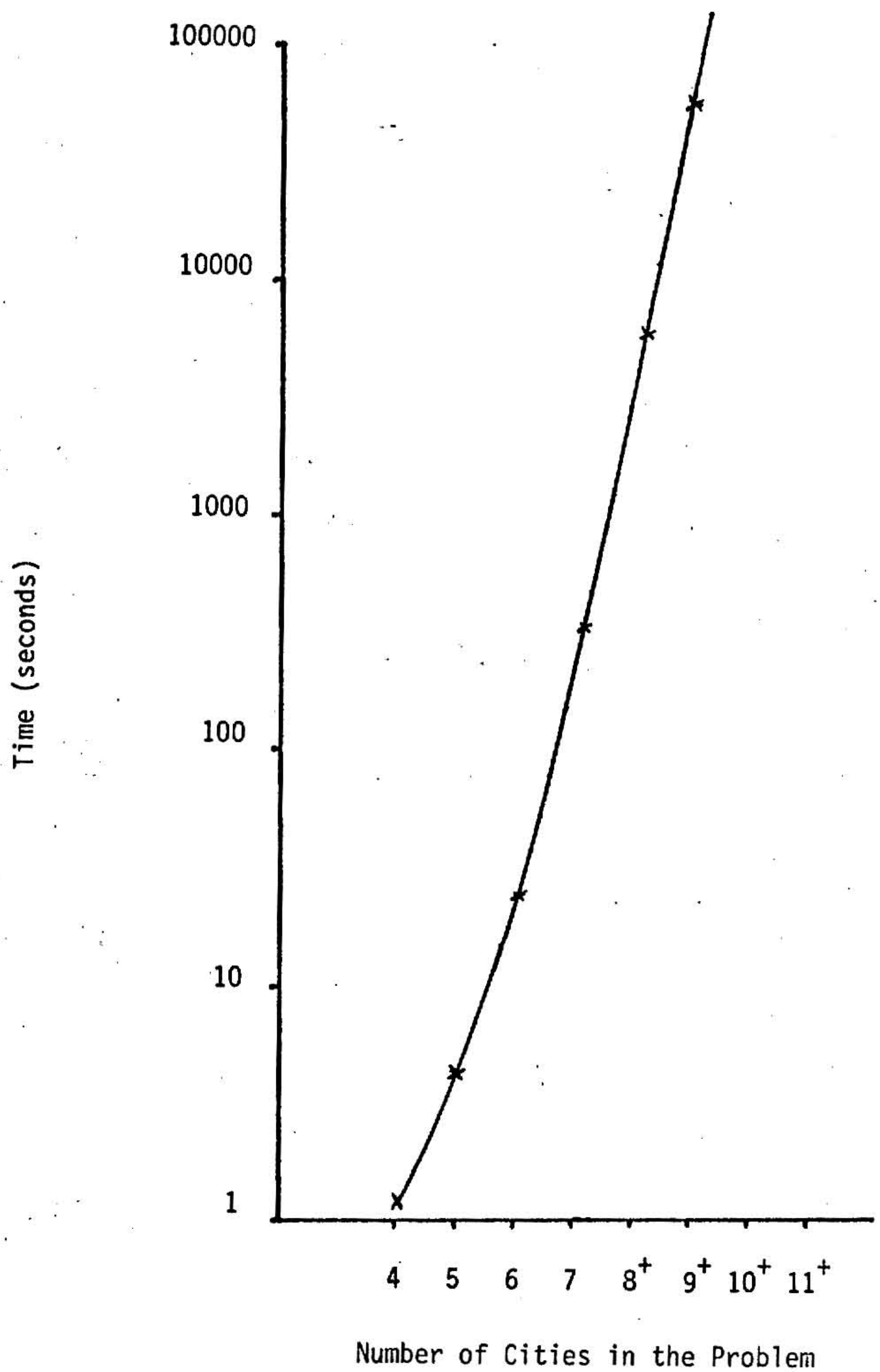
An attempt was made to solve the eleven-city milk route problem employing RIP 30C on an IBM 370/158 computer. No solution was found in twenty minutes of computation time. Subsequent application of RIP 30C to problems of smaller size (i.e. four to eight cities) was attempted. To facilitate adaptation of RIP 30C to the TS problem, a special program was developed which would generate the constraints, for any problem from four to twenty-five cities in size, in a form suitable for input to RIP 30C. (See appendix A.)

A time versus size curve, made by plotting known size versus solution time points and then extrapolating from these, is shown in Figure 2. According to Figure 2, the computation time for an eleven-city problem is well over 100,000 seconds. According to Little, Murty, Sweeney, and Karel [27], the standard deviation of solution times for various problems of a given size should increase as the problem size increases. The minimum possible time for a given problem size could be less than the curve shows as integer programming is notorious for rapidly converging on some problems and taking much longer on others, [6]. However, assuming minimal growth, the time for solution of an eleven-city problem is estimated at over three hours of computation time.

Figure 2

RIP 30C City Size Versus Computation Time*

(Logarithmic Plot)



*Computation time on an IBM 370-158.
+Projected estimates.

An Alternative Approach

In view of the large amount of estimated time in comparison to the problem size, an attempt was made to develop an algorithm designed specifically for traveling salesman problems, in hopes of lowering the solution time into the realm of practicality.

In examining the literature, it was noticed that several sources stated that explicit enumeration of the possible routes was generally impossible because of the large number of possible solutions. In fact, according to Arnoff and Sengupta [2], a twenty-city TS problem on a computer considering 1 tour every microsecond (1/1,000,000 of a second) would take 38,000 years to solve. However, an adaptation of explicit enumeration that only looks at the portion of the routes that differ from a previously generated route, as all possible routes were examined, might allow the solution of small problems in a reasonable amount of time. This would be similar to a procedure discussed by Wells [36], only faster due to savings from only considering the distance associated with links that change from route to route.

The generation of routes in a specific order by changing portions of a previous route falls into the broad category of combinatorial programming. A model for the TS problem, when approached from this perspective, differs from the model given previously for use with RIP 30C. Lin [26] expresses a combinatorial model as follows:

Given cost matrix $D = (d_{ij})$ where d is the cost of
going from city i to city j, 1,j=1,2,...n

Find a permutation $P = (i_1, i_2, \ldots i_n)$ of integers 1 thru n that minimizes the quantity $d_{i_1 i_2} + d_{i_2 i_3} + d_{i_n i_1}$

CITDIS Development

In hopes that time savings would allow solution of a problem at least as large as the eleven-city milk route problem, an algorithm (CITDIS) has been developed. (See appendix B.) It does generate all possible routes in some cases, however, certain shortcuts were incorporated to reduce the number that must be examined in most problems.

The following steps are taken by the algorithm in arriving at the shortest route. For purposes of clarity, the effects of each step on a six-city sample problem are reviewed.

1) CITDIS generates one route at a time in a manner not unlike an odometer. For example, the first route generated in the six-city problem is equal to (1)-2-3-4-5-6-(1), where the numbers stand for cities. (See Figure 3.) The 1 on each end of the route is not generated as this remains constant for every route since each route starts and ends with city 1. The next route is generated by adding one to the next to the last number of the previous route (yielding (1)-2-3-4-6-6-(1)) and then checking it versus all the preceding numbers to see that it equals none of them. This insures that no city is visited twice on the route. Next, the last number in the route previously generated is given the value of 2 (yielding (1)-2-3-4-6-2-(1)) and is incremented until it too is not equal in value to any other number in the route. At this point, the next route would

Figure 3. Routes in the Order of Generation for a Six-City Problem

Column #	1 2 3 4 5		1 2 3 4 5		1 2 3 4 5
1)	2-3-4-5-6	43)	3-6-2-4-5	85)	5-4-2-3-6
2)	2-3-4-6-5	44)	3-6-2-5-4	86)	5-4-2-6-3*
3)	2-3-5-4-6	45)	3-6-4-2-5	87)	5-4-3-2-6
4)	2-3-5-6-4	46)	3-6-4-5-2*	88)	5-4-3-6-2*
5)	2-3-6-4-5	47)	3-6-5-2-4	89)	5-4-6-2-3*
6)	2-3-6-5-4	48)	3-6-5-4-2*	90)	5-4-6-3-2*
			---------		
7)	2-4-3-5-6	49)	4-2-3-5-6	91)	5-6-2-3-4*
8)	2-4-3-6-5	50)	4-2-3-6-5	92)	5-6-2-4-3*
9)	2-4-5-3-6	51)	4-2-5-3-6	93)	5-6-3-2-4*
10)	2-4-5-6-3	52)	4-2-5-6-3*	94)	5-6-3-4-2*
11)	2-4-6-3-5	53)	4-2-6-3-5	95)	5-6-4-2-3*
12)	2-4-6-5-3	54)	4-2-6-5-3*	96)	5-6-4-3-2*
					---------
13)	2-5-3-4-6	55)	4-3-2-5-6	97)	6-2-3-4-5*
14)	2-5-3-6-4	56)	4-3-2-6-5	98)	6-2-3-5-4*
15)	2-5-4-3-6	57)	4-3-5-2-6	99)	6-2-4-3-5*
16)	2-5-4-6-3	58)	4-3-5-6-2*	100)	6-2-4-5-3*
17)	2-5-6-3-4	59)	4-3-6-2-5	101)	6-2-5-3-4*
18)	2-5-6-4-3	60)	4-3-6-5-2*	102)	6-2-5-4-3*
19)	2-6-3-4-5	61)	4-5-2-3-6	103)	6-3-2-4-5*
20)	2-6-3-5-4	62)	4-5-2-6-3*	104)	6-3-2-5-4*
21)	2-6-4-3-5	63)	4-5-3-2-6	105)	6-3-4-2-5*
22)	2-6-4-5-3	64)	4-5-3-6-2*	106)	6-3-4-5-2*
23)	2-6-5-3-4	65)	4-5-6-2-3*	107)	6-3-5-2-4*
24)	2-6-5-4-3	66)	4-5-6-3-2*	108)	6-3-5-4-2*
	---------				
25)	3-2-4-5-6	67)	4-6-2-3-5	109)	6-4-2-3-5*
26)	3-2-4-6-5	68)	4-6-2-5-3*	110)	6-4-2-5-3*
27)	3-2-5-4-6	69)	4-6-3-2-5	111)	6-4-3-2-5*
28)	3-2-5-6-4	70)	4-6-3-5-2*	112)	6-4-3-5-2*
29)	3-2-6-4-5	71)	4-6-5-2-3*	113)	6-4-5-2-3*
30)	3-2-6-5-4	72)	4-6-5-3-2*	114)	6-4-5-3-2*
			---------		
31)	3-4-2-5-6	73)	5-2-3-4-6	115)	6-5-2-3-4*
32)	3-4-2-6-5	74)	5-2-3-6-4*	116)	6-5-2-4-3*
33)	3-4-5-2-6	75)	5-2-4-3-6	117)	6-5-3-2-4*
34)	3-4-5-6-2*	76)	5-2-4-6-3*	118)	6-5-3-4-2*
35)	3-4-6-2-5	77)	5-2-6-3-4*	119)	6-5-4-2-3*
36)	3-4-6-5-2*	78)	5-2-6-4-3*	120)	6-5-4-3-2*
37)	3-5-2-4-6	79)	5-3-2-4-6		
38)	3-5-2-6-4	80)	5-3-2-6-4*		
39)	3-5-4-2-6	81)	5-3-4-2-6		
40)	3-5-4-6-2*	82)	5-3-4-6-2*		
41)	3-5-6-2-4	83)	5-3-6-2-4*		
42)	3-5-6-4-2*	84)	5-3-6-4-2*		

*reverse of a previously generated route be completely generated. (Equal to (1)-2-3-4-6-5-(1).) This process is repeated until the value of the second number from the right (-6- in the second route) is equal to the number of cities in the problem, at which point the next number to the left is incremented one, and the first and second number from the right are lowered to 2, one at a time, and incremented until they do not equal any of the other numbers or themselves. (This would yield the third route of 2-3-5-4-6.) This process continues to the left until all the routes have been generated. See Figure 3, which shows the routes in the order in which they would be generated for a six-city problem.

According to Hall and Knuth [21], it is faster to generate routes by interchanging adjoining city numbers. For example, a four-city problem would have its routes generated as follows: (1)-2-3-4-(1)
(1)-2-4-3-(1)
(1)-4-2-3-(1)
(1)-4-3-2-(1)
(1)-3-4-2-(1)
(1)-3-2-4-(1)

However, the time savings from not recalculating distances for any of the route except the portion that changes would not be nearly as great as in the pattern generation used by CITDIS, simply because a given pattern would repeat fewer times than with CITDIS's method.

2) After each route is generated, it is first checked to see if it is a reverse of a previously generated route. (For example, (1)-4-3-2-(1) is the reverse of (1)-2-3-4-(1). See Figure 3.) This is because the algorithm is designed for the usual case where $c_{ij} = c_{ji}$. (If this is not true, the algorithm can be adjusted by removing the statements that check

for the reverse routes.) Reverse routes are found by checking if the number of the last city in the route is less than the number of the first city. If this is so, then the route must be the reverse of another since the routes are generated in order and all those starting with a smaller number than the route at hand will already have been generated. (See the starred routes in Figure 3--they are reverse routes.)

As one may note from observing Figure 3, the last group of routes, all those starting with 6 (for a six-city problem), are reverses of previously generated routes. This has been found to hold true in general and because of this, the algorithm has been adjusted to not generate this last group of reverse routes as they are already known to be reverses. Thus, the route generation is reduced by 1/n-1 (1/5 in this case). In addition, checking for reverses reduces the generated routes whose distance must be computed (the non-reverse routes) from (n-1)! to (n-1)!/2 (or sixty in the six-city case).

3) As one can also note by examining Figure 3, the way the routes are generated causes a pattern to develop in the repetition of city numbers within the routes. For example, in Figure 3, the first city number (column 1) repeats in groups of twenty-four, the next (column 2) in groups of six, the next (column 3) in groups of two, and the last two columns (columns 4 and 5) do not repeat. This was found to hold true in general with the number of times a given city occurs in a route position before changing being given by

$r_{s-1} \; t_s = r_s$ where r_{s-1} = the number of times the route position to the right of position r_s repeats

(starting with the third position to the right as the initial r_s)

and $t_s = 2,3,\ldots n-2$

s = column index

While various portions of the routes remain constant for a number of repetitions, so would the distance associated with those portions.

Thus, if one kept track of where in the repetitive patterns the route generation was, it would only be necessary to calculate the distance on the portion of the routes that did change and add this to the portion that did not change to achieve the total route distance with fewer computations. CITDIS does just that in that the distance for each link of the route that changes is added one link at a time to the portion that does not change. For example, consider route 74 in Figure 3. Since the distance for the portion of route 74 containing links 1-5-2 would already have been determined for route 73, the distance is not recalculated for this part of the route, but only the rest of the route that does differ from route 73. (2-3-6-4-1) (One may note that city 3 is in the same column position in both routes, and should not its distance from city 2 stay the same and thus not have to be recalculated? The distance does stay the same, but it takes more computation to keep track of the pattern position of a column whose city numbers only repeat twice than it does to recalculate the distance.)

4) As each link, comprising a route's distance, is added, the shortest distance to that point is compared to the shortest distance found thus far for a complete route. If the distance at that point (the

addition of a given link) exceeds the shortest distance found so far for a total route, then the whole route obviously exceeds the shortest distance found so far, as no negative distances are allowed. If this occurs, the route under consideration is rejected along with all other routes in the same pattern group having the same links up to that point.

For example, consider the routes for a six-city problem, shown in Figure 3. Suppose that route number 73's distance was being checked. If the distance of the first link, 1-5 (not shown in Figure 3 as links on front and back of pattern to city 1 are obvious), exceeded the distance of the shortest total route found thus far, then the following twenty-three routes would be skipped. This is because they would have to also exceed the distance of the shortest total route thus far found as they all contain link 1-5 which has been shown to be longer. Other routes, not in the next twenty-three routes, also contain the link 1-5, but they are not immediately rejected because exactly where they will occur is not predictable (with computational efficiency).

If the route causing the distance to be too large was one position to the right in route 73 (link 5-2, columns 1 and 2--links with city 1 are not shown in Figure 3), then only five other routes would be skipped as six is the number of times the link combination 1-5-2 is repeated for a problem of that size.

Should the link being added not cause the route to that point to exceed the shortest route thus far found (the distance of which is initially set to 9999.0 miles, although a lower figure can be input if an upper bound on the problem is known), then the next link is added. The total distance with the next link's distance added is then tested. This sequence is repeated until the route is either rejected, based on its total distance or a portion thereof, or else its distance is as short or shorter than the best distance for a route found previously. If it is shorter, a record of the links in the route(s) that up until then was the shortest is erased and the new shortest route links are stored along with its distance. This distance becomes the new standard for comparison against which all subsequent routes are checked to see if they are shorter. If the route presently being considered is equal in distance to the previous shortest route(s), then the links are stored without erasing those of the previous shortest route(s). Thus, if a problem has more than one best (shortest) route, the algorithm will give them all as answers.

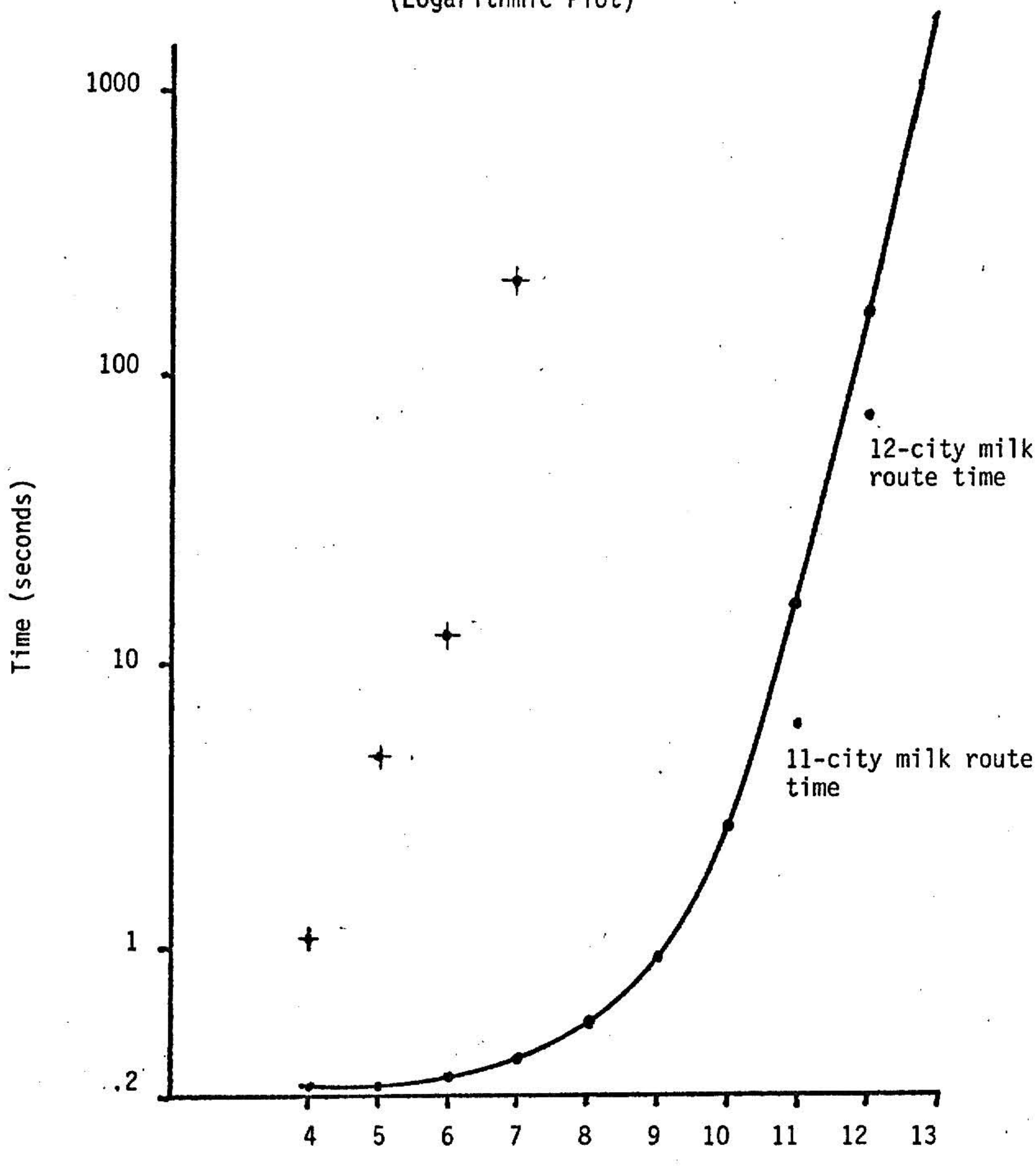
5) If, in the adding of distances associated with links in a given route, a link with a distance of infinity is encountered, it indicates that there is no link between the cities associated with that link. As in step 4) above, CITDIS then skips all routes in the same repetitive pattern with the same link in the same position. (In practice, zeros are used to represent the distance of infinity to the algorithm.) As each link is added, a no-link condition is tested for at the same time the total distance to that point (see the previous step) is checked.

Given the way CITDIS functions, solutions can be found quicker if the cities were renumbered with the city having the fewest links with other cities being given the number 1 and the city with the most links being given the number n. This is because the city with the fewest links to it, if numbered 1 will be encountered first, and thus cause many other routes to be deleted due to the effect of no-links on the route patterns. Fewer of the routes generated at first will turn out to be reverses of previous routes. (Examination of Figure 3 will confirm this.) Because of this, if a no-link is encountered early in the route generating, many routes can be deleted that would not otherwise have been (due to being found to be a reverse of another route).

The CITDIS algorithm was tested on various sized problems. The solution times (for a FORTRAN IV version), the resultant time versus size curve, and the RIP 30C times for the same problems (where applicable) are shown in Figure 4. It can be seen that the CITDIS algorithm is several times faster than the RIP 30C algorithm. This was expected, for although RIP 30C examines the solutions implicitly, the number of 0 - 1 combinations that must be examined either implicitly or explicitly are 2^r, where r is the number of variables. The CITDIS algorithm, although it examines routes more explicitly, must examine only $(n-1)!/2$ routes. For the milk route problem, this amounts to 2^{98} (10^{30}) solutions for RIP 30C versus 1,814,000 (10^6) that CITDIS must consider.

Figure 4

CITDIS City Size Versus Computation Time*
(Logarithmic Plot)



*Computation time on an IBM 370-158.
+RIP 30C time for the same size problem.

IV. RESULTS

CITDIS found the shortest route to be 107.5 miles in length. This compares favorably with the current route of 116.5 miles. The two routes, the one now and the shortest possible, are listed in Table 2.

The nine-mile (7.7%) savings represented by the shorter route may not seem to be of major importance, yet this represents a savings of approximately 468 gallons of gasoline per year (assuming a five day week and six miles per gallon of gas). This amounts to about $234 per year (at $.50 per gallon) plus savings in oil and vehicle wear. These are the savings from applying the traveling salesman analysis to only one truck route. Consider the savings if this was multiplied by the twenty plus routes the dairy has or if we look at a larger application such as the bus routing problem solved approximately by Angel, Noonan, and Winston, [1].

The 2300 or so less miles driven per year, should the shorter route be implemented, would allow the truck driver to possibly include more cities in his route. If similiar savings could be gleaned on other routes, it would most likely allow consolidation of milk routes and thus the use of fewer trucks. For example, a city (Lyndon), was selected for hypothetical inclusion in the eleven-city route.

The problem was solved via CITDIS to yield the inclusion of the Lyndon stop between the Vassar-Hedgewood Acres stop and the Osage City stop to achieve the shortest possible route of 111.5 miles which is five

Table 2

Comparison of Original and Shortest Route

	Original Route*			Shortest Route*	
1)[1]	Topeka		1)	Topeka	
		(17.5mi.)			(17.5mi.)
2)	Auburn		2)	Auburn	
		(16.5mi.)			(11.0mi.)
3)	Carbondale		5)	Burlingame	
		(9.5mi.)			(9.0mi.)
4)	Overbrook		6)	Osage City	
		(8.0mi.)			(10.5mi.)
5)	Michigan Valley		7)	Vassar-Hedgewood Acres	
		(9.0mi.)			(3.0mi.)
6)	Pomona		11)	Green Acres	
		(7.5mi.)			(7.5mi.)
7)	Green Acres		10)	Pomona	
		(3.0mi.)			(9.0mi.)
8)	Vassar-Hedgewood Acres		9)	Michigan Valley	
		(10.5mi.)			(8.0mi.)
9)	Osage City		8)	Overbrook	
		(9.0mi.)			(10.0mi.)
10)	Burlingame		4)	Scranton	
		(7.0mi.)			(5.5mi.)
11)	Scranton		3)	Carbondale	
		(19.0mi.)[2]			(16.5mi.)
1)	Topeka		1)	Topeka	
		116.5 miles			107.5 miles

* Cities are in the order they occur on the routes. Mileage between them is given in parentheses.

[1]These numbers are the ones standing for the city in the cost-link matrix (see Table 1).

[2]This link is not included in the cost-link matrix as cities on either side of Scranton preclude going directly from Scranton to Topeka from being on the shortest route.

miles shorter than the original eleven-city route. (See Table 3 for the associated cost-link matrix.) In the same manner, CITDIS or similiar algorithms could be used to determine which of several cities to add to the route, or which city to delete and what to replace it with, and so on.

Table 3

Cost-Link Matrix for the Twelve-City Milk Route

(Miles Between Cities)

← To City →

Original City Numbers ↓ →		(2)	(7)	(1)	(5)	(10)	(12)	(4)	(9)	(11)	(3)	(6)	(8)
	x	1*	2	3	4	5	6	7	8	9	10	11	12
(2)	1	∞	∞	17.5	∞	11.0	∞	∞	∞	∞	13.5	∞	∞
(7)	2	∞	∞	∞	4.5	∞	∞	∞	∞	∞	∞	7.5	3.0
(1)	3	17.5	∞	∞	26.0	26.0	∞	∞	∞	∞	16.5	∞	∞
(5)	4	∞	4.5	26.0	∞	∞	12.5	8.0	∞	∞	∞	9.0	7.5
(10)	5	11.0	∞	26.0	∞	∞	∞	∞	9.0	7.0	12.5	∞	∞
(12)	6	∞	∞	∞	12.5	∞	∞	19.0	9.5	15.0	∞	16.0	5.0
(4)	7	∞	∞	∞	8.0	∞	19.0	∞	24.5	10.0	9.5	17.0	15.5
(9)	8	∞	∞	∞	∞	9.0	9.5	24.5	∞	20.5	20.0	25.0	10.5
(11)	9	∞	∞	∞	∞	7.0	15.0	10.0	20.5	∞	5.5	30.0	16.0
(3)	10	13.5	∞	16.5	∞	12.5	∞	9.5	20.0	5.5	∞	30.5	15.5
(6)	11	∞	7.5	∞	9.0	∞	16.0	17.0	25.0	30.0	30.5	∞	10.5
(8)	12	∞	3.0	∞	7.5	∞	5.0	15.5	10.5	16.0	15.5	10.5	∞

↑ From City ↓

*Cities were renumbered for input on basis of number of links to a given city. The upper numbers are the original city numbers (12 = Lyndon) where the numbers match those used elsewhere in this report.

V. CONCLUSION

One would assume that, since the new route found results in a cost savings of nine miles per trip, the route should be implemented immediately. This may, however, not be the best solution. One must consider the detrimental effects of switching to a new, shorter route. The new roads used in it could be of differing quality than the old or busier and thus take longer to traverse. This was not the case in this particular problem, but if it were, the difficulty could be overcome by changing the costs, in miles, of the links to a composite figure representing the relative values of the links in terms of miles and time of traverse.

Other problems, in terms of customer and driver adjustment to new route times could occur, although one would expect these effects to be temporary. (This may not be so in larger cases such as multi-truck routing. Therein introduction of better routes would tend to make the dispatcher appear incompetent.)

The above problems serve to point out one advantage of solving a problem of this sort via integer programming--in that the time loss on routes, times of pick up, etc. can be included as constraints. The best CITDIS can do would be to arrange the input costs as discussed above or print out all the routes below a certain distance in length and then subjectively select the one meeting the distance, time, and other criteria best.

CITDIS Improvements

CITDIS could also be modified in other ways. For example, by switching the sections that test for the shortest route, the algorithm could be converted to one that solves the longest path problem. This would be slightly less efficient in that routes could not be rejected for exceeding the upper bound. However, checking the distance of the number of links remaining in a route times the longest possible link's distance against a lower bound could be used to reject routes.

CITDIS could also be modified to print out every feasible route considered that met certain distance criterion and thus provide alternative choices of routes in case other factors cause the shortest route not to be optimal.

TS Analysis Benifits

The problem to which traveling salesman analysis was applied here is but a small example of the many potential traveling salesman type problems. One can also use the traveling salesman type of analysis to minimize cost or time for a route, depending on which values are used in the cost-link matrix. For example, solution of bus routing problems involves solving a TS problem, [6]. TS analysis has also been applied to system state analysis[24], job sequence scheduling [32], and machine scheduling, [16,20,10]. TS analysis, in terms of M-salesman problems, has been applied to many truck dispatching problems, [9,14,7].

Not all the procedures used return the exact best possible routing or scheduling, however, even a slight improvement over the existing situation can result in great cost savings. The optimal solution in a real world situation is not always the shortest route due to biases and inaccuracies in determining the constraints of the model. The only rule governing the use of TS analysis is that the benefits outweigh the costs incurred in determining and using a new problem solution. In these times of increasing energy shortages and cost-consciousness, the future use of TS analysis cannot help but to increase.

VI. LITERATURE CITED

1) Angel, R.D., Caudle, W.L., Noonan, R., and Whinston, A., "Computer Assisted School Bus Scheduling," Management Science, Vol. 18, No. 6 (1972), pp.B279-288.

2) Arnoff, E.L. and Sengupta, S.S., "The Traveling Salesman Problem," Progress in Operations Research, Vol. I (R.L. Ackoff, ed.) Wiley, New York, 1961, pp.150-157.

3) Balas, Egon, "An Additive Algorithm for Solving Linear Programs with Zero-One Variables," Operations Research, Vol. 13 (1965), pp.517-542.

4) Barachet, L.L., "Graphic Solution of the Traveling Salesman Problem," Operations Research, Vol. 5 (1957), pp. 841-845.

5) Bellmore, Mandell, and Malone, John, "Pathology of Traveling-Salesman Subtour-Elimination Algorithms," Operations Research, Vol. 19 (1971), pp. 278-307.

6) Bellmore, M., and Nemhauser, G., "The Traveling Salesman Problem: A Survey," Operations Research, Vol. 16 (1968), pp. 538-558.

7) Christofides, N., and Eilon, S., "An Algorithm for the Vehicle Dispatching Problem," Operations Research Quarterly, Vol. 20 (1969), pp. 309-318.

8) Christofides, Nicos, and Eilon, Samuel, "Algorithms for Large-scale Traveling Salesman Problems," Operations Research Quarterly, Vol. 23, No. 4 (1972), pp. 511-518.

9) Clarke, G., and Wright, J.W., "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points," Operations Research, Vol. 12, (1964), pp. 563-581.

10) Conway, Richard, Maxwell, William, and Miller, Louis, Theory of Scheduling, Addison-Wesley Publishing Co., Reading, Massachusetts, 1967.

11) Croes, G.A., "A Method For Solving Traveling-Salesman Problems," Operations Research, Vol. 6, No. 6 (1958), pp. 791-812.

12) Dantzig, G., Fulkerson, R., and Johnson, S., "Solution of a Large-Scale Traveling-Salesman Problem," Journal of the Operations Research Society of America, Vol. 2, No. 4 (1954), pp. 393-410.

13) Dantzig, G.B., Fulkerson, D.R., and Johnson, S.M., "On a Linear-Programming, Combinatorial Approach to the Traveling-Salesman Problem," Operations Research, Vol. 7, No. 7 (1959), pp. 58-70.

14) Dantzig, G.B., and Ramser, J.H., "The Truck Dispatching Problem," Management Science, Vol. 6 (1960), pp. 80-91.

15) Fleischman, Bernhard, "Computational Experience with the Algorithm of Balas," Operations Research, Vol. 15 (1967), pp. 153-155.

16) Flood, M.H., "The Traveling Salesman Problem," Operations Research, Vol. 4, No. 1 (1956), pp. 61-75.

17) Freeman, R.J., "Computational Experience with the Balas Integer Programming Algorithm," The Rand Corporation, P-3241, Oct. 1965.

18) Geoffrion, A.M., and Nelson, A.B., "User's Instructions for 0-1 Linear Programming Code RIP 30C," Rand Research Memorandum, RM-5627-PR, May 1968.

19) Geoffrion, A.M., "An Improved Implicit Enumeration Approach for Integer Programming," Operations Research, Vol. 17, No. 3 (1969), pp. 437-454.

20) Gilmore, P.C., and Gomory, R.E., "Sequencing a One State-Variable Machine: A Solvable Case of the Traveling Salesman Problem," Operations Research, Vol. 12 (1964), pp. 655-679.

21) Hall, Marshall, and Knuth, D.E., "Combinatorial Analysis and Computers," American Mathmatical Monthly, Vol. 72 (1965), pp. 21-28.

22) Held, Michael, and Karp, Richard M., "The Traveling-Salesman Problem and Minimum Spanning Trees," Operations Research, Vol. 18 (1970), pp. 1138-1162.

23) Hillier, F., and Lieberman, G., Introduction to Operations Research, Holden-Day, Inc., San Francisco, 1967.

24) Hu, T.C., Proceedings of the Fourth International Conference on Operations Research, (Hertz, David B., and Melese, Jacques, eds.), Wiley-Interscience, New York, 1966, pp.1021-1027.

25) Karg, Robert L., and Thompson, Gerald L., "A Heuristic Approach to Solving Traveling Salesman Problems," Management Science, Vol. 10, No. 2 (1964), pp. 225-248.

26) Lin, Shen, "Computer Solutions of the Traveling Salesman Problem," Bell System Technical Journal, Vol. 44 (1965), pp.2245-2269.

27) Little, John, Murty, D.C., Katta, G., Sweeney, Dura A., and Karel, Caroline, "An algorithm for the Traveling Salesman Problem," Operations Research, Vol. 11 (1963), pp. 972-989.

28) Miller, C.E., Tucker, A.W., and Zemlin, R.A., "Integer Programming Formulations and Traveling Salesman Problems," J. ACM, Vol. 7 (1960), pp. 326-329.

29) Nicholson, T.A.J., "A Boundary Method for Planar Traveling Salesman Problems," Operations Research Quarterly, Vol. 19 (1968), pp. 445-451.

30) Plane, Donald R., and Kochenberger, Gary H., Operations Research for Managerial Decisions, Richard D. Irwin, Inc., Homewood, Illinois, 1972.

31) Richmond, Samual B., Operations Research for Management Decisions, The Ronald Press Co., New York, 1968.

32) Rothkopf, Michael, "A Traveling Salesman Problem: On the Reduction of Certain Large Problems to Smaller Ones," Operations Research, Vol. 14 (1966), pp. 532-533.

33) Svestka, Joseph A., and Huckfeldt, Vaughn E., "Computational Experience with an M-Salesman Traveling Salesman Algorithm," Management Science, Vol. 19, No. 7 (1973), pp. 790-799.

34) Wagner, H.M., Principles of Operations Research with Applications, Prentice-Hall, Inc., New Jersey, 1969.

35) Webb, M.H.J., "Some Methods of Producing Approximate Solutions to Traveling Salesman Problems with Hundreds or Thousands of Cities," Operations Research, Vol. 22 (1971), pp. 49-66.

36) Wells, Mark, Elements of Combinatorial Programming, Pergamon Press, New York, 1971.

VII. APPENDIX

Appendix A. PAUL I

This program generates the equations for a traveling salesman problem(s), up to twenty-five cities in size, necessary for Rip 30C to solve the problem. Output consists of: 1) a cost-link matrix reflecting the values input to PAUL I; 2) printing out the step by step construction of the problem constraints and their binary expansions; and 3) punched cards containing the equations in the format necessary for input into Rip 30C.

The input necessary for PAUL I consists of the problem cost-link matrix and the number of cities in the problem. The first card should contain the number of cities in columns 1-2. The next n cards contain the matrix. It is input a row at a time with negative numbers or zero for the no-link positions (where two cities are not directly connected). Each number, including those at the end of a card, must be separated from the next by a comma or a blank. The distances input can be up to 999 miles in length and as accurate as one place below the decimal point.

```
TURNER:  PRCCEDURE CPTICNS(MAIN)RECRDER;
-/*                                                                     */
 /*  VARIABLES                                                          */
 /*                                                                     */
 /*      BI - THE B(I) VECTOR                                           */
 /*                                                                     */
 /*      BIGCNT - A CCUNTER PCINTING TC THE LAST ELEMENT ENTERED IN     */
 /*               BIGMATRIX                                             */
 /*                                                                     */
 /*      BIGGY - A BINARY FIXED (15) VARIABLE USED FOR PASSING THE      */
 /*              VALUE 32767 TO SCAN AND ENTER                          */
 /*                                                                     */
 /*      BIGMATRIX - A STRUCTURE CONTAINING THE ELEMENTS OF THE         */
 /*                  CCNSTRAINT MATRIX.  IT CONSISTS OF A ROW NUMBER,   */
 /*                  A COLUMN NUMBER, AND THE MATRIX ELEMENT THAT       */
 /*                  BELONGS IN THAT LOCATION.                          */
 /*                                                                     */
 /*      CARD_BUF - A TEMPORARY BUFFER USED TC STCRE VALUES BEFORE      */
 /*                 THEY ARE PUNCHED TO CARDS.                          */
 /*                                                                     */
 /*      CARD_CNT - A COUNTER PCINTING TO THE LAST NUMBER ENTERED       */
 /*                 INTC CARD_BUF                                       */
 /*                                                                     */
 /*      COLHDRS - A VECTOR CONTAINING THE HEADERS FOR THE CONSTRAINT   */
 /*                MATRIX                                               */
 /*                                                                     */
 /*      DIM - THE DIMENSION OF THE BETWEEN CITY DISTANCE MATRIX        */
 /*                                                                     */
 /*      EQUATE1      CHARACTER STRINGS USED TOR STCRING THE            */
 /*      ECUATE2      EQUATICNS THAT ARE GENERATED BY THE               */
 /*      EQUATE3      PROGRAM.  THEY ARE ALSO USED AT VARIOUS           */
 /*      EQUATE4      POINTS FOR GENERATING HEADINGS.                   */
 /*      EQUATE5                                                        */
 /*      EQUATE6                                                        8?
 /*                                                                     */
 /*      I,J,K,L,M - LCCP INDICES                                       */
 /*                                                                     */
 /*      MATRIX - THE BETWEEN CITY DISTANCE MATRIX                      */
 /*                                                                     */
 /*      CLDNUM       THE VALUES OF THE RCW AND NUMBER OF BIGMATRIX     */
 /*      CLDRCW       THE LAST TIME THROUGH THE MATRIX PRINT LINE       */
 /*                   GENERATICN LCCP                                   */
 /*                                                                     */
 /*      PRINT_LINE - A CHARACTER STRING USED FOR GENERATION OF         */
 /*                   PRINT LINES                                       */
 /*                                                                     */
 /*      PRTLINE - THE CHARACTER STRING USED FCR GENERATICN OF THE      */
 /*                CONSTRAINT MATRIX PRINT LINE                         */
 /*                                                                     */
 /*      RCODE - THE RETURN CODE FRCM SORT/MERGE                        */
 /*                                                                     */
 /*      RECCNT - A CCUNT OF THE LINES OF PRINT GENERATED FOR THE       */
 /*               CONSTRAINT MATRIX                                     */
 /*                                                                     */
 /*      SIZE - THE AMCUNT OF STCRAGE TO BE USED BY SCRT/MERGE          */
 /*                                                                     */
 /*      SUM - USED TO COLLECT THE COEFFICIENTS FCR THE BINARY          */
 /*            EXPANSICN FORM OF THE EQUATICNS.                         */
 /*                                                                     */
 /*      TEMP - USED TO PASS COLUMN NUMBERS BETWEEN THE SUBROUTINES     */
 /*                                                                     */
 /*      TR       PICTURE VARIABLES USED IN EQUATION GENERATION         */
 /*      TR1                                                            */
-DECLARE MATRIX(25,25) FLOAT DECIMAL (6) CCNTRCLLED,
         DIM FIXED BINARY (15),
         PRINT_LINE CHARACTER(132),
         CARD_BUF(6) FLCAT DECIMAL(6),
         CARD_CNT FIXED BINARY (15),
         TEMP FIXED BINARY (15),
         BIGGY FIXED BINARY (15) INITIAL (32767),
         EQUATE1 CHARACTER (1500) VARYING,
         EQUATE2 CHARACTER (1500) VARYING,
         EQUATE3 CHARACTER(3100) VARYING,
         EQUATE4 CHARACTER(500) VARYING,
         EQUATE5 CHARACTER(500) VARYING,
         EQUATE6 CHARACTER(500) VARYING,
         SUM FIXED BINARY(15),
         (I,J,K,L,M) FIXED BINARY (15),
         ROWCNT FIXED BINARY (15),
         TRI CHARACTER (5),
```

```
          TR PICTURE'Z9';
-DECLARE (SYSPRINT,
          PRINT1,
          PRINT2,
          PRINT3,
          PRINT4,
          PRINT5,
          PRINT6,
          PRINT7,
          PRINT8) FILE STREAM PRINT OUTPUT;
-DECLARE PRINTER FILE RECORD;
-DECLARE DUMP FILE RECORD;
-DECLARE 1 BIGMATRIX (1600),
          2 ROW FIXED BINARY (15),
          2 COL FIXED BINARY (15),
          2 NUMBER FLOAT DECIMAL (6);
-DECLARE 1 COLHDRS (1176) CONTROLLED,
          2 ROW FIXED BINARY (15),
          2 COL FIXED BINARY (15);
-DECLARE BI (1176) FIXED BINARY (15) CONTROLLED,
          PRTLINE CHARACTER (32000) CONTROLLED,
          STR CHARACTER(8),
          HDRCNT FIXED BINARY (15) INITIAL (0),
          BIGCNT FIXED BINARY (31),
          X FIXED BINARY (15),
          OLDROW FIXED BINARY (15),
          SIZE FIXED BINARY (31) INITIAL (100000),
          RCODE FIXED BINARY (31),
          RECCNT FIXED BINARY(15);
-DECLARE IHESARC ENTRY (FIXED BINARY (31)),
          IHESRTD ENTRY (CHARACTER(33),CHARACTER(25),FIXED BINARY(31),
               FIXED BINARY(31),ENTRY,ENTRY);
1/*  CHECKER CHECKS TO SEE IF BIGMATRIX IS FULL, AND IF IT IS,          */
 /*  IT IS WRITTEN TO DUMP, AND BIGCNT IS SET TO ZERO.                  */
-CHECKER:  PROCEDURE;
 IF BIGCNT < 1600 THEN RETURN;
 WRITE FILE (DUMP) FROM (BIGMATRIX);
 BIGCNT=0;
 RETURN;
 END CHECKER;
1/*  ENTER PLACES A Y(I) HEADER INTO THE COLUMN HEADER ARRAY.  THE      */
 /*  ROW NUMBER IS SET TO 32767 SO THAT THE CORRESPONDING BIGMATRIX     */
 /*  ENTRIES WILL APPEAR IN THE PROPER LOCATION AFTER SORTING.          */
 /*  THE HEADER COUNT IS INCREMENTED TO REFLECT THE ADDITION.           */
 /*                                                                     */
 /*  VARIABLES USED                                                     */
 /*                                                                     */
 /*       J - AN INPUT PARAMETER CONTAINING THE Y INDEX TO BE ENTERED   */
 /*                                                                     */
 /*       X - AN OUTPUT PARAMETER CONTAINING THE ACTUAL COLUMN NUMBER   */
 /*           WHERE THE INDEX IS STORED.                                */
 /*                                                                     */
-ENTER:  PROCEDURE (J,X);
-DECLARE (J,X) FIXED BINARY (15);
/HDRCNT=HDRCNT+1;
 COLHDRS.ROW(HDRCNT)=32767;
 COLHDRS.COL(HDRCNT)=J;
 X=HDRCNT;
 RETURN;
 END ENTER;
1/*  SCAN PERFORMS A BINARY SEARCH OF THE COLUMN HEADERS.               */
 /*                                                                     */
 /*                                                                     */
 /*  VARIABLES USED                                                     */
 /*                                                                     */
 /*       I,J - INPUT PARAMETERS CONTAINING THE ROW AND COLUMN          */
 /*             NUMBERS, RESPECTIVELY, TO BE SEARCHED FOR.              */
 /*                                                                     */
 /*       X - THE OUTPUT PARAMETER.  IT CONTAINS THE CONSTRAINT MATRIX */
 /*           COLUMN IF THE I,J COMBINATION IS FOUND, OTHERWISE IT     */
 /*           CONTAINS ZERO.                                            */
 /*                                                                     */
 /*       TOP - THE TOP OF THE CURRENT BINARY SEARCH RANGE              */
 /*                                                                     */
 /*       MID - POINTS TO THE ELEMENT CURRENTLY BEING CHECKED           */
 /*                                                                     */
 /*       BOT - THE BOTTOM OF THE CURRENT BINARY SEARCH RANGE           */
 /*                                                                     */
 /*       SCANNUM - THE SUM OF (I*DIM) AND J.  IT IS USED AS THE        */
 /*                 SEARCH ARGUMENT IN THE BINARY SEARCH.               */
```

```
/*                                                                      */
/*       CURRNUM - THE CURRENT ELEMENT BEING EXAMINED.  IT IS           */
/*                 COMPUTED IN THE SAME WAY AS SCANNUM.                 */
/*                                                                      */
-SCAN:  PROCEDURE (I,J,X);
-DECLARE (I,J,X,TOP,MID,BOT) FIXED BINARY (15),
         (SCANNUM,CURRNUM) FIXED BINARY (31);
-TOP=HDRCNT;
 BOT=1;
 SCANNUM=(PRECISION(DIM,31,0)*PRECISION(I,31,0))+J;
 DO WHILE (TOP>=BOT);
     MID=(TOP+BOT)/2;
     CURRNUM=(PRECISION(DIM,31,0)*PRECISION(COLHDRS.ROW(MID),31,0))+
          COLHDRS.COL(MID);
     IF SCANNUM = CURRNUM THEN DO;
          X=MID;
          RETURN;
     END;
     IF TOP=BOT THEN TOP=TOP-2;
     ELSE IF SCANNUM>CURRNUM THEN BOT=MID+1;
     ELSE TOP=MID-1;
 END;
 X=0;
 RETURN;
 END SCAN;
1/*  INPROC PROVIDES THE INPUT DATA, TAKEN FROM BIGMATRIX, TO THE       */
 /*  SORT/MERGE PROGRAM.                                                */
 /*                                                                     */
 /*                                                                     */
 /*  VARIABLES USED                                                     */
 /*                                                                     */
 /*       MATRIXCNT - A STATIC VARIABLE USED TO POINT TO THE NEXT       */
 /*                   ELEMENT OF BIGMATRIX TO BE PASSED TO SORT         */
 /*                                                                     */
 /*       REC - A CHARACTER STRING USED TO PASS THE DATA TO SORT        */
 /*                                                                     */
 /*       FLAG - A BIT USED TO INDICATE THAT NO MORE DATA WILL BE       */
 /*              PASSED TO SORT ON THE NEXT INVOCATION OF INPROC        */
-INPROC:  PROCEDURE RETURNS (CHARACTER(18));
-DECLARE MATRIXCNT FIXED BINARY (15) INITIAL (1601) STATIC,
         REC CHARACTER (18) INITIAL (' ') STATIC,
         FLAG BIT (1) INITIAL ('0'B) STATIC;
-/*  IF THE CURRENT VALUE OF MATRIXCNT INDICATES THAT THE CURRENT        */
 /*  BLOCK OF DATA IN BIGMATRIX HAS ALL BEEN PASSED TO SORT, A NEW      */
 /*  BLOCK IS READ OFF OF DUMP, AND THE MATRIX COUNTER IS RESET TO 1    */
-IF MATRIXCNT = 1601 THEN DO;
     READ FILE (DUMP) INTO (BIGMATRIX);
     MATRIXCNT = 1;
 END;
-/*  IF FLAG WAS SET ON IN THE PREVIOUS INVOCATION OF THIS ROUTINE,      */
 /*  A RETURN CODE OF 8 IS PASSED BACK TO SORT, VIA IHESARC,            */
 /*  INDICATING THAT THERE IS NO MORE DATA TO BE PASSED.                */
-IF FLAG THEN CALL IHESARC (8);
-/*  WHEN A RECORD IS PASSED, IT IS MOVED, VIA UNSPEC, TO THE FIRST     */
 /*  8 BYTES OF REC.  A CHECK IS MADE TO SEE IF THIS ELEMENT IS THE     */
 /*  LAST ONE, AND IF SO, FLAG IS SET.  THE MATRIX COUNTER IS THEN      */
 /*  INCREMENTED AND A RETURN CODE OS 12 IS PASSED TO SORT,             */
 /*  INDICATING THAT ANOTHER CALL IS TO BE MADE TO THIS ROUTINE.        */
-ELSE DO;
    UNSPEC (STR) = UNSPEC (BIGMATRIX.ROW(MATRIXCNT))    ||
                   UNSPEC (BIGMATRIX.COL(MATRIXCNT))    ||
                   UNSPEC (BIGMATRIX.NUMBER(MATRIXCNT));
    SUBSTR(REC,1,8) = STR;
    IF BIGMATRIX.NUMBER(MATRIXCNT) = -32000 THEN FLAG = '1'B;
    MATRIXCNT=MATRIXCNT+1;
    CALL IHESARC (12);
    RETURN (REC);
 END;
 END INPROC;
1/*  OUTPROC ACCEPTS OUTPUT RECORDS FROM SORT AND REFORMATS THEM SO     */
 /*  THAT THEY MAY BE PLACED INTO BIGMATRIX.                            */
 /*                                                                     */
 /*                                                                     */
 /*  VARIABLES USED                                                     */
 /*                                                                     */
 /*       INREC - A CHARACTER STRING USED TO ACCEPT RECORDS FROM SORT   */
 /*                                                                     */
 /*       LAST - A POINTER TO THE LOCATION IN BIGMATRIX IN WHICH THE    */
 /*              REFORMATTED RECORD IS TO BE PLACED                     */
 /*                                                                     */
```

```
 /*      FLAG - A BIT THAT IS INITALLY SET CN TO INDICATE THAT THE       */
 /*             FILE DUMP HAS NOT HAD ITS STATUS CHANGED FROM INPUT      */
 /*             TO OUTPUT                                                */
-OUTPROC:  PROCEDURE (INREC);
-DECLARE INREC CHARACTER (18),
         LAST FIXED BINARY (15) STATIC INITIAL (1) ,
         FLAG BIT (1) STATIC INITIAL ('1'B);
-/*  IF FLAG IS SET (INDICATING THAT THIS IS THE FIRST ENTRY INTO THE */
 /*  RCUTINE), DUMP IS CLCSED AND RE-OPENED FOR CUTPUT.                */
-IF FLAG THEN DO;
     CLOSE FILE (DUMP);
     OPEN FILE (DUMP) CUTPUT;
     PUT FILE (SYSPUNCH) SKIP;
     FLAG = '0'B;
 END;
-/*  THE RECCRD PASSED BACK FROM SORT IS THEN MOVED, USING UNSPEC,    */
 /*  TO ITS PROPER LCCATION IN BIGMATRIX, AND LAST IS INCREMENTED.    */
-STR=SUBSTR(INREC,1,8);
 BEGIN;
     DECLARE STR1 CHARACTER (8);
     DECLARE STR2 CHARACTER(8);
     DECLARE STR3 CHARACTER (8);
     STR1=SUBSTR(STR,1,2);
     UNSPEC(BIGMATRIX.ROW(LAST))=UNSPEC(STR1);
     STR2=SUBSTR(STR,3,2);
     UNSPEC(BIGMATRIX.COL(LAST))=UNSPEC(STR2);
     STR3=SUBSTR(STR,5,4);
     UNSPEC(BIGMATRIX.NUMBER(LAST))=UNSPEC(STR3);
 END;
 IF BIGMATRIX.NUMBER(LAST)¬=-32000 THEN
 PUT FILE (SYSPUNCH) EDIT (BIGMATRIX(LAST))(F(3),F(3),F(11,1));
 PUT FILE (SYSPUNCH) EDIT (' ')(A);
 IF MOD(LAST,4) = 0 THEN PUT FILE (SYSPUNCH) SKIP;
 LAST=LAST+1;
-/*  IF LAST IS 1601, INDICATING THAT BIGMATRIX HAS BEEN FILLED, OR   */
 /*  THE NUMBER PREVICUSLY RETURNED FRCM SORT IS -32000, INDICATING   */
 /*  THE END OF THE DATA, THE CURRENT CONTENTS OF BIGMATRIX ARE       */
 /*  WRITTEN TC DUMP, AND LAST IS SET TO 1.  A RETURN CODE OF 4,      */
 /*  WHICH REQUESTS ANOTHER RECORD, IS THEN SENT TC SORT VIA IHESARC  */
-IF LAST=1601 | BIGMATRIX.NUMBER(LAST-1)=-32000 THEN DO;
     WRITE FILE (DUMP) FROM (BIGMATRIX);
     LAST=1;
 END;
 CALL IHESARC(4);
 END CUTPROC;
1/*  THE PROGRAM BEGINS BY OPENING ALL FILES.  THE PRINT FILES ARE    */
 /*  OPENED WITH A LINESIZE OF 132 CHARACTERS TO ALLOW MAXIMUM USAGE  */
 /*  OF THE PAGE WIDTH.  THE PROGRAM TERMINATES WHEN AN END OF FILE   */
 /*  IS ENCCUNTERED ON SYSIN.                                         */
-OPEN FILE (DUMP) OUTPUT;
 OPEN FILE (SYSPRINT) LINESIZE(132),
       FILE(PRINT1) LINESIZE(132),
       FILE(PRINT2) LINESIZE(132),
       FILE(PRINT3) LINESIZE(132),
       FILE(PRINT4) LINESIZE(132),
       FILE(PRINT5) LINESIZE(132),
       FILE(PRINT6) LINESIZE(132),
       FILE(PRINT7) LINESIZE(132),
       FILE(PRINT8) LINESIZE(132);
 ON ENDFILE (SYSIN) STOP;
-/*  NOTE:  THE PROGRAM WILL ALLOW FOR MULTIPLE BETWEEN CITY DISTANCE */
 /*  MATRICES TO BE PROCESSED IN A GIVEN RUN.  IT IS NOT RECOMMENDED  */
 /*  THAT THIS BE DONE TOO OFTEN, AS THE CUTPUT FOR THE DIFFERENT     */
 /*  SETS OF DATA IS INTERSPERSED IN SUCH A WAY AS TO REQUIRE A LOT   */
 /*  OF TIME TC BE SPENT BY THE USER SEPERATING THE OUTPUT.           */
 /*                                                                   */
 /*                                                                   */
 /*  THE DIMENSION OF THE MATRIX TC BE PROCESSED IS READ IN AND       */
 /*  CHECKED TC SEE IF IT IS IN THE ALLOWABLE BOUNDS.  IF IT IS NOT,  */
 /*  PROCESSING IS TERMINATED.  IF IT DOES FALL WITHIN LIMITS, THE    */
 /*  COLHDR, BI, AND MATRIX VARIABLES ARE ALLOCATED STORAGE, AND      */
 /*  THE BETWEEN CITY DISTANCE MATRIX IS READ IN.                     */
-DO WHILE ('1'B);
     GET FILE(SYSIN) EDIT (DIM) (COL(1),F(2));
     IF DIM > 25 THEN DO;
               PUT SKIP EDIT ('DIMENSION EXCEEDS ALLOWABLE MAXIMUM.',
                   ' PROCESSING TERMINATED.')(A,A);
               STOP;
            END;
     ALLOCATE COLHDRS ((DIM-1)*(2*DIM-1)),
```

```
    BI ((DIM-1)*(2*DIM-1)),
    MATRIX(DIM,DIM) INITIAL ((DIM*DIM) (0.0));
DO I = 1 TO DIM;
    DO J = 1 TO DIM;
        IF I ¬= J THEN GET FILE (SYSIN) LIST (MATRIX(I,J));
    END;
END;
/*  AFTER THE READING OF THE DATA IS COMPLETE, THE TITLES          */
/*  NEEDED FOR PRINTING THE BETWEEN CITY DISTANCE MATRIX ARE       */
/*  GENERATED.  SINCE THE MAXIMUM DIMENSION ALLOWED IS 25, THE     */
/*  TITLING IS DONE IN SUCH A WAY AS TO FORCE A 25 X 25 MATRIX     */
/*  TO PRINT ON EXACTLY ONE PAGE.  ALL OF THE TITLING IS DONE      */
/*  USING VARYING LENGTH CHARACTER STRINGS TO ENSURE THAT THERE    */
/*  IS ONLY AS MUCH TITLING AS IS NEEDED FOR A GIVEN DIMENSION.    */
PUT FILE(SYSPRINT) EDIT ('BETWEEN CITY DISTANCE MATRIX')
    (PAGE,COL(52),A);
EQUATE1 = 'CITY  ';
DO I = 1 TO DIM;
    TR = I;
    EQUATE1 = EQUATE1 || '   ' || TR || ' ';
END;
PUT FILE (SYSPRINT) SKIP (3) EDIT (EQUATE1)(A);
EQUATE1 = '      ';
DO I = 1 TO DIM;
    EQUATE1 = EQUATE1 || '_____';
END;
PUT FILE (SYSPRINT) SKIP (0) EDIT (EQUATE1)(A);
/*  THE NUMBERS IN THE MATRIX ARE PRINTED IN THE FOLLOWING         */
/*  MANNER:  IF THE NUMBER IS GREATER THAN ZERO, IT WILL BE        */
/*  PRINTED.  IN PLACE OF ALL DIAGONAL ELEMENTS, AN ASTERISK       */
/*  IS PRINTED.  IF THE NUMBER IS LESS THAN ZERO, 'NL' IS PRINTED*/
DO I = 1 TO DIM;
    TR = I;
    EQUATE1 = ' ' || TR || '  |';
    DO J = 1 TO DIM;
        IF I = J THEN EQUATE1 = EQUATE1 || '    * ';
        ELSE IF MATRIX (I,J) <= 0.0 THEN EQUATE1 = EQUATE1 ||
            '   NL';
        ELSE DO;
            PUT STRING (TR1) EDIT (MATRIX(I,J))(F(5,0));
            EQUATE1 = EQUATE1 || TR1;
        END;
    END;
    PUT FILE (SYSPRINT) SKIP EDIT ('|',EQUATE1)
        (COL(6),A,SKIP,A);
END;
/*  NEXT, THE MATRIX, MINUS ANY NUMBERS LESS THAN OR EQUAL TO      */
/*  ZERO, IS PUNCHED TO CARDS, AND ENTERED INTO THE COLUMN         */
/*  HEADER VECTOR.                                                 */
DO I = 1 TO DIM;
    DO J = 1 TO DIM;
        IF MATRIX(I,J) > 0.0 THEN DO;
            HDRCNT=HDRCNT+1;
            COLHDRS.ROW(HDRCNT)=I;
            COLHDRS.COL(HDRCNT)=J;
            CARD_CNT = CARD_CNT + 1;
            CARD_BUF(CARD_CNT) = MATRIX(I,J);
            IF CARD_CNT = 6 THEN DO;
                PUT FILE (SYSPUNCH) SKIP EDIT (CARD_BUF)
                    (F(11,1),X(1));
                CARD_CNT = 0;
            END;
        END;
    END;
END;
PUT FILE (SYSPUNCH) SKIP EDIT ((CARD_BUF(I) DO I = 1 TO CARD_CNT))
    (F(11,1),X(1));
/*  THE FIRST STEP IN THE GENERATION OF EQUATIONS IS TO PRINT      */
/*  THE TITLES AT THE TOP OF THE PAGE.                             */
PUT FILE (SYSPRINT) EDIT ('ROW CONSTRAINTS')(PAGE,COL(59),A);
PUT FILE (PRINT1) EDIT ('ROW CONSTRAINTS')(PAGE,COL(59),A);
PUT FILE (PRINT2) EDIT ('ROW CONSTRAINTS')(PAGE,COL(59),A);
PUT FILE (PRINT3) EDIT ('COLUMN CONSTRAINTS')(PAGE,COL(57),A);
PUT FILE (PRINT4) EDIT ('COLUMN CONSTRAINTS')(PAGE,COL(57),A);
PUT FILE (PRINT5) EDIT ('COLUMN CONSTRAINTS')(PAGE,COL(57),A);
/*  AT THE BEGINNING OF EACH ITERATION OF THE OUTER LOOP,          */
/*  COLUMN OR ROW HEADERS ARE PRINTED ON SYSPRINT AND PRINT3,      */
/*  AND THE EQUATION CHARACTER STRINGS ARE NULLED.                 */
DO I = 1 TO DIM;
    PUT FILE (SYSPRINT) EDIT ('ROW',I) (SKIP(3),A,F(3));
```

```
PUT FILE (PRINT3) EDIT ('CCLUMN',I)(SKIP(3),A,F(3));
EQUATE1,EQUATE2,EQUATE3,EQUATE4,EQUATE5,EQUATE6 = '';
/*  IN THE INNER LOOPS, THE EQUATIONS ARE GENERATED BY          */
/*  SUMPLY CONCATENATING VARIOUS CHARACTER STRINGS              */
/*  TOGETHER.  AS EACH ROW ELEMENT IS ADDED TO THE              */
/*  EQUATIONS, THE ELEMENT IS ADDED TO BIGMATRIX, USING SCAN */
/*  TO FIND THE PROPER COLUMN FOR THE NUMBER TO BE PLACED IN.*/
/*  THE VALUES IN THE BI VECTOR ARE ALSO ENTERED AT THIS     */
/*  TIME.                                                     */
DO J = 1 TO DIM;
    IF MATRIX(I,J) > 0.0 THEN DO;
        TR = I;
        EQUATE1 = EQUATE1 || 'X(' || TR || ',';
        EQUATE2 = EQUATE2 || 'X(' || TR || ',';
        EQUATE3 = EQUATE3 || 'X(' || TR || ',';
        TR = J;
        EQUATE1 = EQUATE1 || TR || ') + ';
        EQUATE2 = EQUATE2 || TR || ') + ';
        EQUATE3 = EQUATE3 || TR || ') - ';
        CALL SCAN(I,J,X);
        BIGCNT=BIGCNT+1;
        BIGMATRIX.ROW(BIGCNT)=(I*2)-1;
        BIGMATRIX.COL(BIGCNT)=X;
        BIGMATRIX.NUMBER(BIGCNT)=-1;
        CALL CHECKER;
        BIGCNT=BIGCNT+1;
        BIGMATRIX.ROW(BIGCNT)=I*2;
        BIGMATRIX.COL(BIGCNT)=X;
        BIGMATRIX.NUMBER(BIGCNT)=1;
        CALL CHECKER;
        BI(I*2-1)=1;
        BI(I*2)=-1;
    END;
    IF MATRIX(J,I) > 0.0 THEN DO;
        TR = J;
        EQUATE4 = EQUATE4 || 'X(' || TR || ',';
        EQUATE5 = EQUATE5 || 'X(' || TR || ',';
        EQUATE6 = EQUATE6 || 'X(' || TR || ',';
        TR = I;
        EQUATE4 = EQUATE4 || TR || ') + ';
        EQUATE5 = EQUATE5 || TR || ') + ';
        EQUATE6 = EQUATE6 || TR || ') - ';
        CALL SCAN (J,I,X);
        BIGCNT=BIGCNT+1;
        BIGMATRIX.ROW(BIGCNT)=(2*DIM)+(I*2)-1;
        BIGMATRIX.COL(BIGCNT)=X;
        BIGMATRIX.NUMBER(BIGCNT)=-1;
        CALL CHECKER;
        BIGCNT=BIGCNT+1;
        BIGMATRIX.ROW(BIGCNT)=(2*DIM)+(I*2);
        BIGMATRIX.COL(BIGCNT)=X;
        BIGMATRIX.NUMBER(BIGCNT)=1;
        CALL CHECKER;
        BI((2*DIM)+(I*2)-1)=1;
        BI((2*DIM)+(I*2))=-1;
    END;
END;
/*  AN ATTEMPT IS MADE TO PRINT A GIVEN EQUATION ONLY IF IT  */
/*  EXISTS, I.E. ITS LENGTH IS GREATER THAN ZERO.  SINCE THE */
/*  EQUATIONS CAN EXCEED ONE PRINT LINE IN LENGTH, THEY      */
/*  ARE CHECKED FOR LENGTH, AND IF NECESSARY, ARE SPLIT      */
/*  IN HALF AND PRINTED ON TWO LINES.                         */
/*  THIS PROCESS APPLIES FOR ALL 6 EQUATIONS GENERATED IN    */
/*  THE ABOVE SECTION.                                        */
IF LENGTH(EQUATE1) ¬= 0 THEN DO;
    EQUATE1 = SUBSTR(EQUATE1,1,LENGTH(EQUATE1)-2);
    IF LENGTH (EQUATE1)< 121 THEN PUT FILE (SYSPRINT) SKIP EDIT
        (EQUATE1,' < 1',EQUATE1,' > 1')(A,A,SKIP,A,A);
    ELSE DO;
        PUT FILE (SYSPRINT) SKIP EDIT (SUBSTR(EQUATE1,1,132))
            (A);
        PUT FILE (SYSPRINT) SKIP EDIT (SUBSTR(EQUATE1,133),
            ' < 1')(X(10),A,A);
        PUT FILE (SYSPRINT) SKIP EDIT (SUBSTR(EQUATE1,1,132))
            (A);
        PUT FILE (SYSPRINT) SKIP EDIT (SUBSTR(EQUATE1,133),
            ' < 1')(X(10),A,A);
        END;
END;
IF LENGTH (EQUATE2) ¬= 0 THEN DO;
```

```
        EQUATE2 = SUBSTR(EQUATE2,1,LENGTH(EQUATE2)-2);
        IF LENGTH (EQUATE2) < 121 THEN PUT FILE (PRINT1) SKIP EDIT
            ('-1 + ',EQUATE2,' > 0')(A,A,A);
        ELSE DO;
            PUT FILE (PRINT1) SKIP EDIT ('-1 + ',SUBSTR(EQUATE2,
                1,120))(A,A);
            PUT FILE (PRINT1) SKIP EDIT (SUBSTR(EQUATE2,121),' > 0'
                )(X(5),A,A);
            END;
    END;
    IF LENGTH (EQUATE3) ¬= 0 THEN DO;
        EQUATE3 = SUBSTR(EQUATE3,1,LENGTH(EQUATE3)-2);
        IF LENGTH (EQUATE3) < 121 THEN PUT FILE (PRINT2) SKIP EDIT
            ('1 - ',EQUATE3,' > 0')(A,A,A);
        ELSE DO;
            PUT FILE (PRINT2) SKIP EDIT ('1 - ',SUBSTR(EQUATE3,1,
                120))(A,A);
            PUT FILE (PRINT2) SKIP EDIT (SUBSTR(EQUATE3,121),
                ' > 0')(X(5),A,A);
        END;
    END;
    IF LENGTH (EQUATE4) ¬= 0 THEN DO;
        EQUATE4 = SUBSTR(EQUATE4,1,LENGTH(EQUATE4)-2);
        IF LENGTH (EQUATE4) < 121 THEN PUT FILE (PRINT3) SKIP EDIT
            (EQUATE4,' < 1',EQUATE4,' > 1')(A,A,SKIP,A,A);
        ELSE DO;
            PUT FILE (PRINT3) SKIP EDIT (SUBSTR(EQUATE4,1,132))(A);
            PUT FILE (PRINT3) SKIP EDIT (SUBSTR(EQUATE4,133),
                ' < 1')(X(5),A,A);
            PUT FILE (PRINT3) SKIP EDIT (SUBSTR(EQUATE4,1,132))(A);
            PUT FILE (PRINT3) SKIP EDIT (SUBSTR(EQUATE4,133),
                ' < 1')(X(5),A,A);
        END;
    END;
    IF LENGTH (EQUATE5) ¬= 0 THEN DO;
        EQUATE5 = SUBSTR(EQUATE5,1,LENGTH(EQUATE5)-2);
        IF LENGTH (EQUATE5) < 121 THEN PUT FILE (PRINT4) SKIP EDIT
            ('-1 + ',EQUATE5,' > 0')(A,A,A);
        ELSE DO;
            PUT FILE (PRINT4) SKIP EDIT ('-1 + ',SUBSTR(EQUATE5,1,
                120))(A,A);
            PUT FILE (PRINT4) SKIP EDIT (SUBSTR(EQUATE5,121),' > 0'
                )(X(5),A,A);
        END;
    END;
    IF LENGTH (EQUATE6) ¬= 0 THEN DO;
        EQUATE6 = SUBSTR(EQUATE6,1,LENGTH(EQUATE6)-2);
        IF LENGTH (EQUATE6) < 121 THEN PUT FILE (PRINT5) SKIP EDIT
            ('1 - ',EQUATE6,' > 0')(A,A,A);
        ELSE DO;
            PUT FILE (PRINT5) SKIP EDIT ('1 - ',SUBSTR(EQUATE6,1,
                120))(A,A);
            PUT FILE (PRINT5) SKIP EDIT (SUBSTR(EQUATE6,121),
                ' > 0')(X(5),A,A);
        END;
    END;
END;
/*  THE FOLLOWING SECTION GENERATES THE U-SUBTOUR CONSTRAINT       */
/*  EQUATIONS IN THE ORIGIONAL, INTERMEDIATE, AND BINARY           */
/*  EXPANSION FORMS.  THE EQUATION GENERATION TECHNIQUE IS MUCH    */
/*  THE SAME AS IN THE FIRST PART OF THE PROGRAM.                  */
PUT FILE (PRINT6) EDIT ('U-SUBTOUR CONSTRAINTS - ORIGINAL FORM')
    (PAGE,COL(48),A);
PUT FILE (PRINT7) EDIT ('U-CONSTRAINTS - INTERMEDIATE FORM')
    (PAGE,COL(50),A);
PUT FILE (PRINT8) EDIT ('U-CONSTRAINTS - BINARY EXPANSION FORM')
    (PAGE,COL(48),A);
LOG=LOG2(FLOAT(DIM+1))+1;
DO I = 1 TO LOG*(DIM-1);
    COLHDRS.ROW(HDRCNT+I)=32767;
    COLHDRS.COL(HDRCNT+I)=I;
END;
ROWCNT=BIGMATRIX.ROW(BIGCNT);
HDRCNT=HDRCNT+(LOG*(DIM-1));
DO I = 2 TO DIM;
    DO J = 2 TO DIM;
        IF MATRIX(I,J) > 0.0 THEN DO;
            ROWCNT=ROWCNT+1;
            PUT FILE (PRINT6) SKIP EDIT ('U(',I,') - U(',J,') + ',
                DIM,'X(',I,',',J,') < ',DIM-1)(6 (A,F(2)));
```

```
PUT FILE (PRINT7) SKIP EDIT (DIM-1,' - U(',I,') + U(',
     J,') - ',DIM,'X(',I,',',J,') > 0')(6 (F(2),A));
PRINT_LINE=REPEAT(' ',131);
CALL SCAN(I,J,X);
BIGCNT=BIGCNT+1;
BIGMATRIX.ROW(BIGCNT)=ROWCNT;
BIGMATRIX.COL(BIGCNT)=X;
BIGMATRIX.NUMBER(BIGCNT)=-DIM;
CALL CHECKER;
SUM = 0;
TR = I;
EQUATE1 = 'U(' || TR || ') = - ';
DO K = 1 TO 100 WHILE (SUM <= DIM);
     TR = 2**(K-1);
     EQUATE1 = EQUATE1 || TR || 'Y(';
     TR = K+(LOG*(I-2));
     EQUATE1 = EQUATE1 || TR || ') - ';
     SUM = SUM + (2**(K-1));
     TEMP = K+(LOG*(I-2));
     CALL SCAN(BIGGY,TEMP,X);
     IF X = 0 THEN CALL ENTER(TEMP,X);
     BIGCNT=BIGCNT+1;
     BIGMATRIX.ROW(BIGCNT)=ROWCNT;
     BIGMATRIX.COL(BIGCNT)=X;
     BIGMATRIX.NUMBER(BIGCNT)=-(2**(K-1));
     CALL CHECKER;
END;
EQUATE1 = SUBSTR(EQUATE1,1,LENGTH(EQUATE1)-2);
TR = J;
EQUATE2 = 'U(' || TR || ') = ';
SUM = 0;
DO L = 1 TO 100 WHILE (SUM <= DIM);
     TR = 2**(L-1);
     EQUATE2 = EQUATE2 || TR || 'Y(';
     TR = L+(LOG*(J-2));
     EQUATE2 = EQUATE2 || TR || ') + ';
     SUM = SUM + (2**(L-1));
     TEMP = L+(LOG*(J-2));
     CALL SCAN (BIGGY,TEMP,X);
     IF X = 0 THEN CALL ENTER (TEMP,X);
     BIGCNT=BIGCNT+1;
     BIGMATRIX.ROW(BIGCNT)=ROWCNT;
     BIGMATRIX.COL(BIGCNT)=X;
     BIGMATRIX.NUMBER(BIGCNT)=2**(L-1);
     CALL CHECKER;
     BI(ROWCNT)=DIM-1;
END;
/*   WHEN THE EQUATIONS ARE PRINTED, ALLOWANCE IS MADE*/
/*   FOR THE EQUATIONS OCCUPYING MORE THAN ONE PRINT   */
/*   LINE.  THE TECHNIQUE HERE IS TO PRINT THE FIRST   */
/*   PART OF THE EQUATION, EVERYTHING UP TO THE LAST   */
/*   PART, AND THEN THE LAST PART.                     */
EQUATE2 = SUBSTR(EQUATE2,1,LENGTH(EQUATE2)-2);
IF LENGTH (EQUATE1) < 128 THEN PUT FILE (PRINT8)
     SKIP(2) EDIT (EQUATE1)(A);
ELSE DO;
     PUT FILE (PRINT8) SKIP (2) EDIT (SUBSTR(EQUATE1,1,
          128))(A);
     DO M = 129 TO LENGTH(EQUATE1) - 120 BY 120;
          PUT FILE (PRINT8) SKIP EDIT (SUBSTR(EQUATE1,M,
               120))(X(10),A);
     END;
     PUT FILE (PRINT8) SKIP EDIT (SUBSTR(EQUATE1,M))
          (X(10),A);
END;
IF LENGTH(EQUATE2) < 128 THEN PUT FILE (PRINT8) SKIP
     EDIT (EQUATE2)(A);
ELSE DO;
     PUT FILE (PRINT8) SKIP EDIT (SUBSTR(EQUATE2,1,
          128))(A);
     DO M = 129 TO LENGTH(EQUATE2) - 120 BY 120;
          PUT FILE (PRINT8) SKIP EDIT (SUBSTR(EQUATE2,M,
               120))(X(10),A);
     END;
     PUT FILE (PRINT8) SKIP EDIT (SUBSTR(EQUATE2,M))
          (X(10),A);
END;
EQUATE1 = SUBSTR(EQUATE1,INDEX(EQUATE1,'-')+2);
EQUATE2 = SUBSTR(EQUATE2,INDEX(EQUATE2,'=')+2);
TR = DIM - 1;
```

```
                    EQUATE3 = TR || ' - ' || EQUATE1 || '     +     ' ||
                       EQUATE2 || '    -     ';
                    TR = DIM;
                    EQUATE3 = EQUATE3 || TR || 'X(';
                    TR = I;
                    EQUATE3 = EQUATE3 || TR || ',';
                    TR = J;
                    EQUATE3 = EQUATE3 || TR || ') > 0';
                    IF LENGTH (EQUATE3) < 125 THEN PUT FILE (PRINT8) SKIP
                       EDIT (EQUATE3)(A);
                    ELSE DO;
                       PUT FILE (PRINT8) SKIP EDIT (SUBSTR(EQUATE3,1,
                          125))(A);
                       DO M = 126 TO LENGTH(EQUATE3) - 120 BY 120;
                          PUT FILE (PRINT8) SKIP EDIT (SUBSTR(EQUATE3,M,
                             120))(X(10),A);
                       END;
                       PUT FILE (PRINT8) SKIP EDIT (SUBSTR(EQUATE3,M))(A);
                    END;
                 END;
              END;
           END;
           /*  AFTER ALL THE EQUATIONS AND CONSTRAINTS ARE GENERATED, A      */
           /*  TRAILER ELEMENT IS PLACED ON BIGMATRIX, AND THE FINAL RECORD */
           /*  IS WRITTEN TO DUMP.  DUMP IS THEN CLOSED AND RE-OPENED FOR   */
           /*  INPUT, SO THAT INPROC CAN PASS RECORDS TO SORT.              */
           BIGCNT=BIGCNT+1;
           BIGMATRIX.ROW(BIGCNT)=32767;
           BIGMATRIX.COL(BIGCNT)=32767;
           BIGMATRIX.NUMBER(BIGCNT)=-32000;
           WRITE FILE (DUMP) FROM (BIGMATRIX);
           CLOSE FILE (DUMP);
           OPEN FILE (DUMP) INPUT;
           CALL IHESRTD (' SORT FIELDS=(1,2,BI,A,3,2,BI,A) ',
              ' RECORD TYPE=F,LENGTH=18 ',SIZE,RCODE,INPROC,OUTPROC);
           /*  AFTER CONTROL RETURNS FROM SORT/MERGE, THE RETURN CODE IS    */
           /*  CHECKED, AND IF IT IS NON-ZERO, FURTHER PROCESSING IS        */
           /*  ABORTED.  OTHERWISE, THE CONSTRAINS MATRIX PRINT LINE IS     */
           /*  ALLOCATED, AND THE PRINTER SPILL FILE AND DUMP ARE OPENED.   */
           IF RCODE ¬= 0 THEN DO;
              PUT FILE (SYSPRINT) SKIP(5) EDIT ('SORT FAILED.  RC = ',RCODE)
                 (A,F(2));
              STOP;
           END;
           ALLOCATE PRTLINE CHARACTER ((HDRCNT+3)*10);
           CLOSE FILE (DUMP);
           OPEN FILE (PRINTER) OUTPUT,
              FILE (DUMP) INPUT;
           /*  THE FIRST FEW CONSTRAINT MATRIX PRINT LINES GENERATED         */
           /*  CONSIST OF COLUMN TITLES.  THESE LINES ARE CREATED, AND      */
           /*  THEN WRITTEN TO THE SPILL FILE.                              */
           PRTLINE=' ';
           DO I = 1 TO HDRCNT WHILE (COLHDRS.ROW(I) ¬= 32767);
              PUT STRING (SUBSTR(PRTLINE,I*10-8,10)) EDIT
                 (MATRIX(COLHDRS.ROW(I),COLHDRS.COL(I)))(X(2),F(4,0));
           END;
           DO J = I TO HDRCNT;
              PUT STRING (SUBSTR(PRTLINE,J*10-8,10)) EDIT (0.0)(X(2),F(4,0));
           END;
           WRITE FILE (PRINTER) FROM (PRTLINE);
           PRTLINE=' ';
           DO I = 1 TO HDRCNT;
              PUT STRING (SUBSTR(PRTLINE,I*10-8)) EDIT (I)(F(6));
           END;
           WRITE FILE (PRINTER) FROM (PRTLINE);
           PRTLINE=' ';
           DO I = 1 TO HDRCNT WHILE (COLHDRS.ROW(I) ¬= 32767);
              PUT STRING (SUBSTR(PRTLINE,I*10-8,10)) EDIT
                 ('X(',COLHDRS.ROW(I),',',COLHDRS.COL(I),')')
                 (A,F(2),A,F(2),A);
           END;
           DO J = I TO HDRCNT;
              PUT STRING (SUBSTR(PRTLINE,J*10-8,10)) EDIT ('Y(',
                 COLHDRS.COL(J),')')(X(2),A,F(2),A);
           END;
           PUT STRING (SUBSTR(PRTLINE,(LENGTH(PRTLINE)-10),10)) EDIT ('B(I)')
              (X(6),A);
           WRITE FILE (PRINTER) FROM (PRTLINE);
           PRTLINE=SUBSTR(REPEAT('-',32000),1,LENGTH(PRTLINE))
           WRITE FILE (PRINTER) FROM (PRTLINE);
```

```
PRTLINE=' ';
RECCNT=4;
CLDROW=C;
/*  EACH INDIVIDUAL PRINT LINE OF THE CONSTRAINT MATRIX IS        */
/*  GENERATED BASED ON THE SORTED BIGMATRIX.  WHEN A ROW IS       */
/*  COMPLETED, ITS B(I) VALUE IS WRITTEN ON THE END OF THE LINE,  */
/*  AND THE ENTIRE LINE IS WRITTEN TO THE PRINTER FILE.           */
DO WHILE (OLDNUM ¬= -32000);
   READ FILE (DUMP) INTO (BIGMATRIX);
   DO I = 1 TO 1600 WHILE (OLDNUM ¬= -32000);
      IF OLDROW ¬= BIGMATRIX.ROW(I) THEN DO;
         IF OLDROW ¬= 0 THEN PUT STRING (SUBSTR(PRTLINE,(LENGTH(PRTLINE)
            PRTLINE)-10),10)) EDIT (BI(CLDROW))(F(10,1));
         WRITE FILE (PRINTER) FROM (PRTLINE);
         RECCNT=RECCNT+1;
         PRTLINE=' ';
      END;
      CLDROW=BIGMATRIX.ROW(I);
      OLDNUM=BIGMATRIX.NUMBER(I);
      IF OLDNUM ¬= -32000 THEN PUT STRING (SUBSTR(PRTLINE,
         ((BIGMATRIX.COL(I)-1)*10+1),10)) EDIT (BIGMATRIX.NUMBER
         (I))(F(8,1));
   END;
END;
CLOSE FILE (PRINTER);
OPEN FILE (PRINTER);
/*  THE PRINTER SPILL FILE IS THEN READ BACK IN, AND THE LINES    */
/*  OF PRINT ARE SPLIT INTO 130 BYTE WIDTHS.  THESE ARE THEN      */
/*  WRITTEN TO THE REAL PRINTER A "COLUMN" AT A TIME.             */
PUT FILE (PRINT8) PAGE;
READ FILE (PRINTER) INTO (PRTLINE);
PUT FILE (PRINT8) SKIP (2) EDIT ('C(J)',SUBSTR(PRTLINE,1,120))
   (A,X(5),A);
READ FILE (PRINTER) INTO (PRTLINE);
PUT FILE (PRINT8) SKIP (2) EDIT ('A(I,J)',SUBSTR(PRTLINE,1,120))
   (A,X(3),A);
DO I = 1 TO 3;
   READ FILE (PRINTER) INTO (PRTLINE);
   PUT FILE (PRINT8)SKIP(2) EDIT (SUBSTR(PRTLINE,1,120))(X(9),A);
END;
DO I = 1 TO RECCNT-5;
   READ FILE (PRINTER) INTO (PRTLINE);
   PUT FILE(PRINT8)  SKIP (2) EDIT (I,'|',SUBSTR(PRTLINE,1,120))
      (F(7),X(1),A,A);
END;
CLOSE FILE (PRINTER);
DO I = 121 TO (LENGTH(PRTLINE)-120) BY 130;
   OPEN FILE (PRINTER) INPUT;
   PUT FILE (PRINT8) PAGE;
   DO J = 1 TO RECCNT;
      READ FILE (PRINTER) INTO (PRTLINE);
      PUT FILE (PRINT8) SKIP (2) EDIT (SUBSTR(PRTLINE,I,130))(A);
   END;
   CLOSE FILE (PRINTER);
END;
OPEN FILE (PRINTER) INPUT;
PUT FILE (PRINT8) PAGE;
   DO J = 1 TO 2;
      READ FILE (PRINTER) INTO (PRTLINE);
      PUT FILE (PRINT8) SKIP (2) EDIT (SUBSTR(PRTLINE,I))(A);
   END;
DO J = 1 TO RECCNT-2;
   READ FILE (PRINTER) INTO (PRTLINE);
   PUT FILE (PRINT8) SKIP (2) EDIT (SUBSTR(PRTLINE,I))(A);
END;
CARD_CNT=0;
DO I = 1 TO RECCNT-5;
   CARD_CNT=CARD_CNT+1;
   CARD_BUF(CARD_CNT)=BI(I);
   IF CARD_CNT = 6 THEN DO;
      PUT FILE (SYSPUNCH) SKIP EDIT (CARD_BUF)(F(11,1),X(1));
      CARD_CNT=0;
   END;
END;
PUT FILE (SYSPUNCH) SKIP EDIT ((CARD_BUF(I) DO I = 1 TO CARD_CNT))
   (F(11,1),X(1));
END TURNER;
```

Appendix B. CITDIS

This program solves traveling salesman problems up to twelve cities in size. Larger sizes can be solved by adding additional sections simiiliar to those contained herein. Output consists of: 1) a cost-link matrix to reflect the data input; 2) number of feasible routes with all city links in; 3) the shortest route(s) in terms of input city numbers; 4) the length(s) of the shortest route(s); and 5) the initial upper bound used (set at 9999.0 if not input).

The data input consists of: the number of cities in the problem; an initial upper bound, if any; a problem title, if any; and the cost-link matrix. More specific format instructions are included in the program listing.

```
C*********************************************************************
C     THIS PROGRAM FINDS THE SHORTEST ROUTE AMONG M CITIES WHERE
C     M IS LESS THAN 13.  THE SHORTEST ROUTE (OR ROUTES, IF MORE
C     THAN ONE ARE EQUALLY SHORT) IS PRINTED, ALONG WITH THE
C     LENGTH OF THE SHORTEST ROUTE (S).
C*********************************************************************
C     THE SHORTEST ROUTE IS FOUND VIA GENERATING ALL THE POSSIBLE
C     ROUTES (VIA IMBEDDED DO LOOPS).  AS EACH ROUTE IS GENERATED,
C     THE DISTANCE OF EACH LINK IN THE ROUTE (WITH A LINK REPRE-
C     SENTING TRAVEL BETWEEN TWO CITIES) IS SUMMED AND THE TOTAL
C     DISTANCE OF THE ROUTE IS FOUND.  THE DISTANCE OF THE SHORT-
C     EST ROUTE FOUND IS ASSIGNED TO CORDIS WHICH REPRESENTS THE
C     BEST ROUTE DISTANCE SO FAR FOUND.  THE ROUTE GIVING THE
C     BEST DISTANCE IS STORED IN MATRIX ANS FOR LATER OUTPUT.
C*********************************************************************
      DIMENSION A(15,15)
      COMMON/FORMAT/FMT
      COMPLEX*16 FMT(4)
      LOGICAL*1 FMTOVL(64),DIGIT(10)/'0','1','2','3','4','5','6','7','8'
     *,'9'/
      EQUIVALENCE(FMT(1),FMTOVL(1))
      REAL N12,N13,N14,N15,N16,N17,N18,N19,N110,N111,N112
      INTEGER K(15),ANS(15,15)
      INTEGER*4 ITITLE(5)
C*********************************************************************
C     THIS SECTION READS IN THE DATA.  THE FIRST CARD READ CONTAINS
C     ONLY M, CORDIS, AND ITITLE, WHERE M= THE NUMBER OF CITIES
C     IN THE PROBLEM (WHERE 4=MINIMUM, 12=MAXIMUM); CORDIS= AN
C     UPPER BOUND ON THE SOLUTION--IF KNOWN, OTHERWISE CORDIS IS
C     TO 9999.0;  ITITLE= A PROBLEM TITLE UP TO 20 CHARACTERS IN
C     LENGTH.  THE SUBSEQUENT CARDS, M IN NUMBER, EACH CONTAIN A
C     ROW OF THE LINK-DISTANCE MATRIX.  (THIS LINK-DISTANCE
C     MATRIX CONTAINS THE DISTANCES FROM ANY GIVEN CITY IN THE
C     PROBLEM TO ALL OTHERS.)  ZERO IS ENTERED FOR LINKS BETWEEN A
C     CITY AND ITSELF (THE MATRIX DIAGONAL) AND THOSE CASES WHERE A
C     CITY IS NOT CONNECTED TO ANOTHER CITY (AT LEAST NOT DIRECTLY.)
C*********************************************************************
 9998 READ(5,1,END=9999) M,CORDIS,ITITLE
    1 FORMAT(I2,F5.1,5A4)
      IF(CORDIS.NE.0.0) GO TO 1001
      CORDIS=9999.0
 1001 UPLIM=CORDIS
 1000 DO 202 I=1,M
  202 READ(5,203) (A(I,J),J=1,M)
  203 FORMAT(15F4.1)
      WRITE(6,75) ITITLE
   75 FORMAT('1',56X,5A4)
C*********************************************************************
C     THIS SUBROUTINE PRINTS THE LINK-DISTANCE MATRIX, FILLING THE
C     DIAGONAL WITH *'S AND REPLACING THE ZEROS REPRESENTING NO
C     LINK BETWEEN TWO CITIES WITH 'NL'.
C*********************************************************************
      CALL PRINT(A,M)
C*********************************************************************
C     THE FOLLOWING FUNCTION CALL RETURNS A VALUE WHICH REPRESENTS
C     THE NUMBER OF POSSIBLE ROUTES FOR A GIVEN SIZED PROBLEM.  THIS
C     ASSUMES ALL LINKS ARE IN AND THAT THE REVERSE OF A ROUTE
C     IS NOT COUNTED AS A POSSIBLE ROUTE.
C*********************************************************************
      ISIZE=NUMFAC(M)
      WRITE(6,100) ISIZE
  100 FORMAT('0','NUMBER OF FEASIBLE ROUTES WITH ALL LINKS IN = ',I15)
      M1=M-1
      M2=M-2
      M3=M-3
      M4=M-4
      M5=M-5
      M6=M-6
      M7=M-7
      M8=M-8
      M9=M-9
      M10=M-10
      M11=M-11
      ISOL=1
      ICT=0
      ICT6=0
      ICT24=0
      ICT120=0
      ICT720=0
      ICT504=0
```

```
      ICT403=0
      ICT362=0
      ICT36M=0
      IRESET=0
C***********************************************************************
C     EVERY TIME A CARD SUCH AS THE FOLLOWING M=2 CARD IS ENCOUNT-
C     ERED, A NEW SECTION (WITH DO LOOP) COMES INTO PLAY.  THUS A 5
C     CITY PROBLEM WOULD ONLY USE THE SECTIONS MARKED M=2,M=3,M=4,
C     AND M=5 TO SOLVE THE PROBLEM.
C*********************************** M=2 *******************************
      N(1)=1
C***********************************************************************
C  THE FOLLOWING STATEMENT CONTROLS THE OUTER LOOP OR, IN OTHER WORDS,
C     THE GENERATION OF THE FIRST CITY IN THE ROUTE.  THIS COUNTER IS
C     SET TWO LESS THAN THE NUMBER OF CITIES.  THIS IS SO THAT THE LAST
C     GROUP OF ROUTES BEGINNING WITH THE LINK FROM CITY 1 TO CITY M ARE
C     NOT GENERATED SINCE THEY ALL ARE REVERSES OF THE PREVIOUSLY
C     GENERATED ROUTES.
C***********************************************************************
      DO 2 I1=1,M2
      N(1)=N(1)+1
C*********************************** M=3 *******************************
      N(2)=1
      DO 3 I2=1,M2
    4 N(2)=N(2)+1
      IF(N(2).EQ.N(1)) GO TO 4
C**************************************M=4 *****************************
   17 N(3)=1
      DO 5 I3=1,M3
    6 N(3)=N(3)+1
      IF(N(3).EQ.N(2)) GO TO 6
      IF(N(3).EQ.N(1)) GO TO 6
C***********************************************************************
C     THE NEXT STATEMENT AND OTHERS LIKE IT IN EACH M= SECTION OF
C     THE PROGRAM CHECK ON THE SIZE OF THE PROBLEM AND SKIP OVER
C     THE SECTIONS THAT COMPUTE AND CHECK THE DISTANCES FOR EACH
C     ROUTE OF SMALLER SIZED PROBLEMS.  THIS IS SO THAT, BY GOING
C     TO THE NEXT SECTION(S),DO LOOP(S) CAN BE ADDED TO ASSIST
C     IN THE ROUTE GENERATION.  THE ROUTES ARE ESSENTIALLY GENER-
C     ATED LIKE AN ODOMETER IN MOTION EXCEPT THAT NO TWO NUMBERS
C     OF THE ODOMETER FACE CAN BE THE SAME, AND THAT THE NUMBER OF
C     FIGURES IS EQUAL TO M MINUS ONE.  (THIS IS BECAUSE CITY ONE
C     IF LEFT OUT OF THE ROUTE GENERATION AS ALL ROUTES START AND
C     END AT CITY ONE.
C***********************************************************************
      IF(M.NE.4) GO TO 18
C***********************************************************************
C     THE FOLLOWING STATEMENT AND OTHERS THROUGHOUT THE PROGRAM THAT
C     ARE PRECEDED BY  *****CHECK REVERSE*****  ARE INSERTED TO CHECK
C     EACH ROUTE AS IT IS GENERATED TO SEE WHETHER OR NOT IT IS THE
C     REVERSE OF A PREVIOUSLY GENERATED ROUTE.  THIS IS ACCOMPLISHED
C     BY CHECKING THE FIRST CITY IN THE ROUTE VERSUS THE LAST CITY
C     (IGNORING CITY 1).  IF THE LAST CITY IS LESS THAN THE FIRST IT IS
C     A REVERSE OF A PREVIOUSLY GENERATED ROUTE SINCE ALL COMBINATIONS
C     STARTING WITH A LOWER NUMBERED CITY WILL HAVE PREVIOUSLY BEEN
C     CONSIDERED IN EARLIER ROUTE GENERATION.
C***********************************************************************
C     *****CHECK REVERSE*****
      IF(N(3).LT.N(1)) GO TO 5
C***********************************************************************
C     THE FOLLOWING SECTION AND SIMILIAR SECTIONS UNDER M=5 AND
C     M=6 CALCULATE THE DISTANCE FOR EACH ROUTE VIA SUMMING THE
C     INDIVIDUAL LINK DISTANCES.  IF, AS THE LINKS ARE BEING SUMMED,
C     THE SUM AT ANY POINT EXCEEDS THE SHORTEST DISTANCE (UPPER
C     BOUND) THUS FAR FOUND, THE ROUTE IS REJECTED AT THAT POINT
C     AND GENERATION OF THE NEXT ONE IS BEGUN.
C     IF A 'NL' IS ENCOUNTERED (A DISTANCE OF 0.0 AS THE MATRIX
C***********************************************************************
C     DIAGONAL IS NOT CONSIDERED) THE ROUTE IS ABANDONED AT THAT
C     POINT AND GENERATION OF THE NEXT IS BEGUN.  IF THE ROUTE IS A
C     VALID ONE AND IS SHORTER THAN THE SHORTEST DISTANCE THUS FAR
C     FOUND, THE ROUTE AND IT'S DISTANCE ARE STORED--ERASING THE
C     PREVIOUS SHORTEST ROUTE AND IT'S ASSOCIATED DISTANCE.  IF THE
C     ROUTE IS EQUAL IN DISTANCE TO THE SHORTEST FOUND, IT IS STORED
C     WITHOUT ERASING THE OTHER SHORTEST ROUTE(S).  THUS, IF A
C     PROBLEM HAS MORE THAN ONE SOLUTION THEY WILL ALL BE OUTPUTTED.
C***********************************************************************
      IPROB=1
      GO TO 95
C*********************************** M=5 *******************************
```

```
   18 N(4)=1
      DO 7 I4=1,M4
    8 N(4)=N(4)+1
      IF(N(4).EQ.N(3)) GO TO 8
      IF(N(4).EQ.N(2)) GO TO 8
      IF(N(4).EQ.N(1)) GO TO 8
      IF(M.NE.5) GO TO 19
C     *****CHECK REVERSE*****
      IF(N(4).LT.N(1)) GO TO 7
      IPROB=2
      GO TO 95
C************************************ M=6 ****************************
   19 N(5)=1
      DO 9 I5=1,M5
   10 N(5)=N(5)+1
      IF(N(5).EQ.N(4)) GO TO 10
      IF(N(5).EQ.N(3)) GO TO 10
      IF(N(5).EQ.N(2)) GO TO 10
      IF(N(5).EQ.N(1)) GO TO 10
      IF(M.NE.6) GO TO 20
C     *****CHECK REVERSE*****
      IF(N(5).LT.N(1)) GO TO 9
      IPROB =3
   95 DIST=0.0
      DO 70 JJ=1,M
      IF(JJ.EQ.1) GO TO 40
      I=N(JJ-1)
      IF(JJ.EQ.M) GO TO 50
      GO TO 60
   40 I=1
   60 J=N(JJ)
   80 IF(A(I,J).EQ.0.0) GO TO (5,7,9)IPROB
      DIST=A(I,J)+DIST
      IF(DIST.GT.CORDIS) GO TO (5,7,9)IPROB
      GO TO 70
   50 J=1
      GO TO 80
   70 CONTINUE
      IF(DIST.GT.CORDIS) GO TO (5,7,9)IPROB
      IF(DIST.LT.CORDIS) GO TO 142
      ISOL=ISOL+1
      GO TO 143
  142 ISOL=1
C************** THIS MATRIX STORES THE SHORTEST ROUTE(S). ************
  143 DO 144 IT=1,M1
  144 ANS(ISOL,IT)=N(IT)
C*********************************************************************
C     THE NEXT STATEMENT ASSIGNS THE SHORTEST DISTANCE FOUND TO A
C     HOLDER VARIABLE CORDIS.
C*********************************************************************
      CORDIS=DIST
  141 GO TO (5,7,9)IPROB
C************************************ M=7 ****************************
   20 N(6)=1
      DO 11 I6=1,M6
   12 N(6)=N(6)+1
      IF(N(6).EQ.N(5)) GO TO 12
      IF(N(6).EQ.N(4)) GO TO 12
      IF(N(6).EQ.N(3)) GO TO 12
      IF(N(6).EQ.N(2)) GO TO 12
      IF(N(6).EQ.N(1)) GO TO 12
      IF(M.NE.7) GO TO 15
C*********************************************************************
C     THE FOLLOWING SECTION AND SIMILIAR ONES UNDER THE REMAINING
C     M= SECTIONS CALCULATE THE DISTANCE FOR EACH ROUTE AND
C     COMPARE IT TO THE SHORTEST DISTANCE ROUTE FOUND PREVIOUSLY.
C*********************************************************************
C     AS WITH THE PREVIOUS SECTIONS UNDER M=6 ETC., THESE SECTIONS
C     ALSO CKECK EACH LINK IN THE ROUTE, AS IT IS SUMMED TO THE
C     OTHERS, FOR ITS BEING A 'NL' OR THE SUM TO THAT POINT EX-
C     CEEDING THE DISTANCE OF THE SHORTEST ROUTE THUS FAR FOUND.
C     IN EITHER CASE THE ROUTE IS REJECTED.  HOWEVER, UNLIKE THE
C     PREVIOUS SECTIONS, MORE THAN ONE ROUTE AT A TIME CAN BE
C     SKIPPED DEPENDING ON WHERE IN THE ROUTE GENERATION THE LINK
C     THAT IS EQUAL TO ZERO IN DISTANCE OR CAUSES THE TOTAL DIS-
C     TANCE OF THE ROUTE TO THAT POINT TO EXCEED CORDIS (THE SHORT-
C     EST ROUTE DISTANCE THUS FAR FOUND) IS AT.
C*********************************************************************
C     COUNTERS AT THE BEGINNING KEEP TRACK OF WHERE IN THE ROUTE
C     GENERATION PATTERN THE PROGRAM IS AT.  WHEN ON OF THE ABOVE
```

```
C      CONDITIONS IS VIOLATED, COUNTERS ARE INCREMENTED TO KEEP
C      TRACK OF WHERE IN THE ROUTE GENERATION PATTERN THE PROGRAM
C      JUMPS TO, AND THEN THE PROGRAM JUMPS TO THE APPROPRIATE DO
C      LOOP--SKIPPING ALL THE ROUTES THAT NEED NOT BE CONSIDERED,
C      AND IRESET IS ADJUSTED ACCORDINGLY.
C***********************************************************************
C      IRESET TELLS THE PROGRAM HOW MUCH OF THE ROUTE TO RECALCU-
C      LATE THE DISTANCE OF EACH TIME.  FOR EXAMPLE:  IF ONE
C      ROUTE EQUALS 1-2-3-4-5-6-7-1 AND THE NEXT EQUALS 1-2-3-4-
C      5-7-6-1, THEN THE ONLY DISTANCE THAT MUST BE CONSIDERED
C      TO TELL WHICH IS SHORTEST IS THE DISTANCE CORRESPONDING
C      TO LINKS 5-7-6-1 AS THE DISTANCE OF THE REST OF THE ROUTE
C      IS THE SMAE AS THAT OF THE FIRST ROUTE.
C***********************************************************************
C      THIS OMISSION OF GENERATING THE TOTAL DISTANCE EACH TIME
C      EFFECTS LARGER PERCENTAGE TIME SAVINGS THE LARGER THE PROB-
C      LEM BEING CONSIDERED.
C***********************************************************************
C******THIS NEXT SECTION INCREMENTS THE PATTERN POSITION COUNTERS.******
  340 IF(ICT.NE.6) GO TO 300
      ICT6=ICT6+1
      ICT=0
      IRESET=14
  300 IF(ICT6.NE.4) GO TO 301
      ICT24=ICT24+1
      ICT6=0
      IRESET=13
  301 IF(ICT24.NE.5) GO TO 410
      ICT120=ICT120+1
      ICT24=0
      IRESET=0
  410 ICT=ICT+1
C      *****CHECK REVERSE*****
      IF(N(6).LT.N(1)) GO TO 11
      DIST=0.0
C***********************************************************************
C      THIS SECTION CHECKS IRESET AND JUMPS OVER THE APPROPRIATE NUM-
C      BER OF LINK-DISTANCE GENERATIONS.
C***********************************************************************
      IF(IRESET.EQ.15) GO TO 333
      IF(IRESET.EQ.14) GO TO 334
      IF(IRESET.EQ.13) GO TO 335
      IRESET=15
C***********************************************************************
C      THIS ECTION ADDS THE LINKS WHILE CHECKING THEM FOR 'NL' AND
C      EXCEEDING CORDIS.
C***********************************************************************
      IF(A(1,N(1)).EQ.0.0) GO TO 304
      N12=A(1,N(1))
      IF(N12.GE.CORDIS) GO TO 304
  335 IF(A(N(1),N(2)).EQ.0.0) GO TO 305
      N13=N12+A(N(1),N(2))
      IF(N13.GE.CORDIS) GO TO 305
  334 IF(A(N(2),N(3)).EQ.0.0) GO TO 306
      N14=N13+A(N(2),N(3))
      IF(N14.GE.CORDIS) GO TO 306
  333 IF(A(N(3),N(4)).EQ.0.0) GO TO 11
      N15=N14+A(N(3),N(4))
      IF(N15.GE.CORDIS) GO TO 11
      IF(A(N(4),N(5)).EQ.0.0) GO TO 11
      N16=N15+A(N(4),N(5))
      IF(N16.GE.CORDIS) GO TO 11
      IF(A(N(5),N(6)).EQ.0.0) GO TO 11
      N17=N16+A(N(5),N(6))
      IF(N17.GE.CORDIS) GO TO 11
      IF(A(N(6),1).EQ.0.0) GO TO 11
      DIST=N17+A(N(6),1)
C***********************************************************************
C      THIS SECTION STORES THE SHORTEST ROUTE AND THE DISTANCE THUS
C      FAR FOUND AND WORKS THE SAME AS THE CORRESPONDING SEC-
C      TIONS UNDER M=4 THROUGH M=6.
C***********************************************************************
      IF(DIST.GT.CORDIS) GO TO 11
      IF(DIST.LT.CORDIS) GO TO 93
      ISOL=ISOL+1
      GO TO 94
   93 ISOL=1
   94 DO 216 IT=1,M1
  216 ANS(ISOL,IT)=N(IT)
      CORDIS=DIST
```

```
      GO TO 11
C***********************************************************************
C     THE FOLLCWING SECTIONS INCREMENT THE CCUNTER THAT CCRRESPONDS
C     TO A GIVEN DO LOOP JUMP AND SETS ALL LCWER CCUNTERS TO ZERC.
C***********************************************************************
  304 ICT120=ICT120+1
      ICT24=0
      ICT6=0
      ICT=0
      IRESET=0
      GO TO 2
  305 ICT24=ICT24+1
      ICT6=0
      ICT=0
      IRESET=13
      GO TO 3
  306 ICT6=ICT6+1
      ICT=0
      IRESET=14
      GO TO 5
C************************************ M=8 *****aaa*********************
   15 N(7)=1
      CC 13 I7=1,M7
   14 N(7)=N(7)+1
      IF(N(7).EQ.N(6)) GO TO 14
      IF(N(7).EQ.N(5)) GO TO 14
      IF(N(7).EQ.N(4)) GO TO 14
      IF(N(7).EQ.N(3)) GO TO 14
      IF(N(7).EQ.N(2)) GO TO 14
      IF(N(7).EQ.N(1)) GO TO 14
      IF(M.NE.8) GO TC 16
      IF(ICT.NE.6) GC TC 26
      ICT6=ICT6+1
      ICT=0
      IRESET=15
   26 IF(ICT6.NE.4) GO TO 27
      ICT24=ICT24+1
      ICT6=0
      IRESET=14
   27 IF(ICT24.NE.5) GC TO 28
      ICT120=ICT120+1
      ICT24=0
      IRESET=13
   28 IF(ICT120.NE.6) GO TO 29
      ICT720=ICT720+1
      ICT120=0
      IRESET=0
   29 ICT=ICT+1
C     *****CHECK REVERSE*****
      IF(N(7).LT.N(1)) GO TO 13
      DIST=0.0
      IF(IRESET.EQ.16) GO TO 31
      IF(IRESET.EQ.15) GO TO 33
      IF(IRESET.EQ.14) GO TC 34
      IF(IRESET.EQ.13) GO TO 35
      IRESET=16
      IF(A(1,N(1)).EQ.0.0) GO TO 36
      N12=A(1,N(1))
      IF(N12.GE.CORDIS) GO TC 36
   35 IF(A(N(1),N(2)).EQ.0.0) GO TO 37
      N13=N12+A(N(1),N(2))
      IF(N13.GE.CORDIS) GC TC 37
   34 IF(A(N(2),N(3)).EC.0.0) GO TO 38
      N14=N13+A(N(2),N(3))
      IF(N14.GE.CORDIS) GO TO 38
   33 IF(A(N(3),N(4)).EC.0.0) GC TO 39
      N15=N14+A(N(3),N(4))
      IF(N15.GE.CORDIS) GO TO 39
   31 IF(A(N(4),N(5)).EQ.0.0) GC TO 13
      N16=N15+A(N(4),N(5))
      IF(N16.GE.CCRDIS) GC TC 13
      IF(A(N(5),N(6)).EQ.0.0) GC TO 13
      N17=N16+A(N(5),N(6))
      IF(N17.GE.CORDIS) GO TO 13
      IF(A(N(6),N(7)).EQ.0.0) GO TC 13
      N18=N17+A(N(6),N(7))
      IF(N18.GE.CCRDIS) GO TC 13
      IF(A(N(7),1).EQ.C.0) GO TO 13
      CIST=N18+A(N(7),1)
      IF(DIST.GT.CORDIS) GO TC 13
```

```
      IF(DIST.LT.CORDIS) GO TO 41
      ISCL=ISCL+1
      GO TO 42
   41 ISCL=1
   42 DO 43 IT=1,M1
   43 ANS(ISCL,IT)=N(IT)
      CORDIS=DIST
      GO TO 13
   36 ICT720=ICT720+1
      ICT120=0
      ICT24=0
      ICT6=0
      ICT=0
      IRESET=0
      GO TO 2
   37 ICT120=ICT120+1
      ICT24=0
      ICT6=0
      ICT=0
      IRESET=13
      GO TO 3
   38 ICT24=ICT24+1
      ICT6=0
      ICT=0
      IRESET=14
      GO TO 5
   39 ICT6=ICT6+1
      ICT=0
      IRESET=15
      GO TO 7
C********************************** M=9 ***************************
   16 N(8)=1
      DO 21 I8=1,M8
   22 N(8)=N(8)+1
      IF(N(8).EQ.N(7)) GO TO 22
      IF(N(8).EQ.N(6)) GO TO 22
      IF(N(8).EQ.N(5)) GO TO 22
      IF(N(8).EQ.N(4)) GO TO 22
      IF(N(8).EQ.N(3)) GO TO 22
      IF(N(8).EQ.N(2)) GO TO 22
      IF(N(8).EQ.N(1)) GO TO 22
      IF(M.NE.9) GO TO 23
      IF(ICT.NE.6) GO TO 44
      ICT6=ICT6+1
      ICT=0
      IRESET=16
   44 IF(ICT6.NE.4) GO TO 45
      ICT24=ICT24+1
      ICT6=0
      IRESET=15
   45 IF(ICT24.NE.5) GO TO 46
      ICT120=ICT120+1
      ICT24=0
      IRESET=14
   46 IF(ICT120.NE.6) GO TO 47
      ICT720=ICT720+1
      ICT120=0
      IRESET=13
   47 IF(ICT720.NE.7) GO TO 48
      ICT504=ICT504+1
      ICT720=0
      IRESET=0
   48 ICT=ICT+1
C     *****CHECK REVERSE*****
      IF(N(8).LT.N(1)) GO TO 21
      DIST=0.0
      IF(IRESET.EQ.17) GO TO 49
      IF(IRESET.EQ.16) GO TO 51
      IF(IRESET.EQ.15) GO TO 52
      IF(IRESET.EQ.14) GO TO 53
      IF(IRESET.EQ.13) GO TO 54
      IRESET=17
      IF(A(1,N(1)).EQ.0.0) GO TO 55
      N12=A(1,N(1))
      IF(N12.GE.CORDIS) GO TO 55
   54 IF(A(N(1),N(2)).EQ.0.0) GO TO 56
      N13=N12+A(N(1),N(2))
      IF(N13.GE.CORDIS) GO TO 56
   53 IF(A(N(2),N(3)).EQ.0.0) GO TO 57
      N14=N13+A(N(2),N(3))
```

```
      IF(N14.GE.CCRDIS) GC TO 57
   52 IF(A(N(3),N(4)).EQ.0.0) GO TC 58
      N15=N14+A(N(3),N(4))
      IF(N15.GE.CCRDIS) GO TC 58
   51 IF(A(N(4),N(5)).EQ.0.0) GO TO 59
      N16=N15+A(N(4),N(5))
      IF(N16.GE.CORDIS) GO TO 59
   49 IF(A(N(5),N(6)).EQ.0.0) GO TC 21
      N17=N16+A(N(5),N(6))
      IF(N17.GE.CORDIS) GO TO 21
      IF(A(N(6),N(7)).EQ.0.0) GO TO 21
      N18=N17+A(N(6),N(7))
      IF(N18.GE.CORDIS) GO TO 21
      IF(A(N(7),N(8)).EQ.0.0) GO TC 21
      N19=N18+A(N(7),N(8))
      IF(N19.GE.CORDIS) GO TO 21
      IF(A(N(8),1).EQ.0.0) GC TO 21
      DIST=N19+A(N(8),1)
      IF(DIST.GT.CORDIS) GO TO 21
      IF(DIST.LT.CORDIS) GC TO 61
      ISCL=ISOL+1
      GO TO 62
   61 ISCL=1
   62 DO 63 IT=1,M1
   63 ANS(ISCL,IT)=N(IT)
      CORDIS=DIST
      GC TC 21
   55 ICT504=ICT504+1
      ICT720=C
      ICT120=0
      ICT24=0
      ICT6=0
      ICT=0
      IRESET=0
      GO TC 2
   56 ICT720=ICT720+1
      ICT120=C
      ICT24=C
      ICT6=0
      ICT=0
      IRESET=13
      GO TO 3
   57 ICT120=ICT120+1
      ICT24=C
      ICT6=0
      ICT=0
      IRESET=14
      GO TO 5
   58 ICT24=ICT24+1
      ICT6=0
      ICT=0
      IRESET=15
      GO TO 7
   59 ICT6=ICT6+1
      ICT=0
      IRESET=16
      GO TO 9
C*************************************** M=10 ***************************
   23 N(9)=1
      DO 24 I9=1,M9
   25 N(9)=N(9)+1
      IF(N(9).EQ.N(8)) GO TC 25
      IF(N(9).EQ.N(7)) GO TC 25
      IF(N(9).EQ.N(6)) GO TO 25
      IF(N(9).EQ.N(5)) GO TC 25
      IF(N(9).EQ.N(4)) GO TO 25
      IF(N(9).EQ.N(3)) GO TO 25
      IF(N(9).EQ.N(2)) GO TC 25
      IF(N(9).EQ.N(1)) GO TO 25
      IF(M.NE.10) GO TC 106
      IF(ICT.NE.6) GC TO 64
      ICT6=ICT6+1
      ICT=0
      IRESET=17
   64 IF(ICT6.NE.4) GC TC 65
      ICT24=ICT24+1
      ICT6=0
      IRESET=16
   65 IF(ICT24.NE.5) GC TO 66
      ICT120=ICT120+1
```

```
      ICT24=0
      IRESET=15
   66 IF(ICT120.NE.6) GO TO 67
      ICT720=ICT720+1
      ICT120=0
      IRESET=14
   67 IF(ICT720.NE.7) GO TO 68
      ICT504=ICT504+1
      ICT720=0
      IRESET=13
   68 IF(ICT504.NE.8) GO TO 69
      ICT403=ICT403+1
      ICT504=0
      IRESET=0
   69 ICT=ICT+1
C     *****CHECK REVERSE*****
      IF(N(9).LT.N(1)) GO TO 24
      DIST=0.0
      IF(IRESET.EQ.18) GO TO 71
      IF(IRESET.EQ.17) GO TO 72
      IF(IRESET.EQ.16) GO TO 73
      IF(IRESET.EQ.15) GO TO 74
      IF(IRESET.EQ.14) GO TO 76
      IF(IRESET.EQ.13) GO TO 77
      IRESET=18
      IF(A(1,N(1)).EQ.0.0) GO TO 78
      N12=A(1,N(1))
      IF(N12.GE.CORDIS) GO TO 78
   77 IF(A(N(1),N(2)).EQ.0.0) GO TO 79
      N13=N12+A(N(1),N(2))
      IF(N13.GE.CORDIS) GO TO 79
   76 IF(A(N(2),N(3)).EQ.0.0) GO TO 81
      N14=N13+A(N(2),N(3))
      IF(N14.GE.CORDIS) GO TO 81
   74 IF(A(N(3),N(4)).EQ.0.0) GO TO 82
      N15=N14+A(N(3),N(4))
      IF(N15.GE.CORDIS) GO TO 82
   73 IF(A(N(4),N(5)).EQ.0.0) GO TO 83
      N16=N15+A(N(4),N(5))
      IF(N16.GE.CORDIS) GO TO 83
   72 IF(A(N(5),N(6)).EQ.0.0) GO TO 84
      N17=N16+A(N(5),N(6))
      IF(N17.GE.CORDIS) GO TO 84
   71 IF(A(N(6),N(7)).EQ.0.0) GO TO 24
      N18=N17+A(N(6),N(7))
      IF(N18.GE.CORDIS) GO TO 24
      IF(A(N(7),N(8)).EQ.0.0) GO TO 24
      N19=N18+A(N(7),N(8))
      IF(N19.GE.CORDIS) GO TO 24
      IF(A(N(8),N(9)).EQ.0.0) GO TO 24
      N110=N19+A(N(8),N(9))
      IF(N110.GE.CORDIS) GO TO 24
      IF(A(N(9),1).EQ.0.0) GO TO 24
      DIST=N110+A(N(9),1)
      IF(DIST.GT.CORDIS) GO TO 24
      IF(DIST.LT.CORDIS) GO TO 85
      ISOL=ISOL+1
      GO TO 86
   85 ISOL=1
   86 DO 87 IT=1,M1
   87 ANS(ISOL,IT)=N(IT)
      CORDIS=DIST
      WRITE(6,660) CORDIS,(N(IT),IT=1,M1)
  660 FORMAT('0',F5.1,2X,9(I3))
      GO TO 24
   78 ICT403=ICT403+1
      ICT504=0
      ICT720=0
      ICT120=0
      ICT24=0
      ICT6=0
      ICT=0
      IRESET=0
      GO TO 2
   79 ICT504=ICT504+1
      ICT720=0
      ICT120=0
      ICT24=0
      ICT6=0
      ICT=0
```

```
      IRESET=13
      GO TO 3
   81 ICT720=ICT720+1
      ICT120=0
      ICT24=0
      ICT6=0
      ICT=0
      IRESET=14
      GO TO 5
   82 ICT120=ICT120+1
      ICT24=0
      ICT6=0
      ICT=0
      IRESET=15
      GO TO 7
   83 ICT24=ICT24+1
      ICT6=0
      ICT=0
      IRESET=16
      GO TO 9
   84 ICT6=ICT6+1
      ICT=0
      IRESET=17
      GO TO 11
C*********************************** M=11 **************************
  106 N(10)=1
      DO 105 I10=1,M10
  107 N(10)=N(10)+1
      IF(N(10).EQ.N(9)) GO TO 107
      IF(N(10).EQ.N(8)) GO TO 107
      IF(N(10).EQ.N(7)) GO TO 107
      IF(N(10).EQ.N(6)) GO TO 107
      IF(N(10).EQ.N(5)) GO TO 107
      IF(N(10).EQ.N(4)) GO TO 107
      IF(N(10).EQ.N(3)) GO TO 107
      IF(N(10).EQ.N(2)) GO TO 107
      IF(N(10).EQ.N(1)) GO TO 107
      IF(M.NE.11) GO TO 88
      IF(ICT.NE.6) GO TO 116
      ICT6=ICT6+1
      ICT=0
      IRESET=18
  116 IF(ICT6.NE.4) GO TO 117
      ICT24=ICT24+1
      ICT6=0
      IRESET=17
  117 IF(ICT24.NE.5) GO TO 118
      ICT120=ICT120+1
      ICT24=0
      IRESET=16
  118 IF(ICT120.NE.6) GO TO 119
      ICT720=ICT720+1
      ICT120=0
      IRESET=15
  119 IF(ICT720.NE.7) GO TO 120
      ICT504=ICT504+1
      ICT720=0
      IRESET=14
  120 IF(ICT504.NE.8) GO TO 121
      ICT403=ICT403+1
      ICT504=0
      IRESET=13
  121 IF(ICT403.NE.9) GO TO 122
      ICT362=ICT362+1
      ICT403=0
      IRESET=0
  122 ICT=ICT+1
C     *****CHECK REVERSE*****
      IF(N(10).LT.N(1)) GO TO 105
      DIST=0.0
      IF(IRESET.EQ.19) GO TO 91
      IF(IRESET.EQ.18) GO TO 92
      IF(IRESET.EQ.17) GO TO 97
      IF(IRESET.EQ.16) GO TO 98
      IF(IRESET.EQ.15) GO TO 99
      IF(IRESET.EQ.14) GO TO 101
      IF(IRESET.EQ.13) GO TO 102
      IRESET=19
      IF(A(1,N(1)).EQ.0.0) GO TO 103
      N12=A(1,N(1))
```

```
      IF(N12.GE.CORDIS) GO TO 103
  102 IF(A(N(1),N(2)).EQ.0.0) GO TO 104
      N13=N12+A(N(1),N(2))
      IF(N13.GE.CORDIS) GO TO 104
  101 IF(A(N(2),N(3)).EQ.0.0) GO TO 108
      N14=N13+A(N(2),N(3))
      IF(N14.GE.CORDIS) GO TO 108
   99 IF(A(N(3),N(4)).EQ.0.0) GO TO 109
      N15=N14+A(N(3),N(4))
      IF(N15.GE.CORDIS) GO TO 109
   98 IF(A(N(4),N(5)).EQ.0.0) GO TO 110
      N16=N15+A(N(4),N(5))
      IF(N16.GE.CORDIS) GO TO 110
   97 IF(A(N(5),N(6)).EQ.0.0) GO TO 111
      N17=N16+A(N(5),N(6))
      IF(N17.GE.CORDIS) GO TO 111
   92 IF(A(N(6),N(7)).EQ.0.0) GO TO 112
      N18=N17+A(N(6),N(7))
      IF(N18.GE.CORDIS) GO TO 112
   91 IF(A(N(7),N(8)).EQ.0.0) GO TO 105
      N19=N18+A(N(7),N(8))
      IF(N19.GE.CORDIS) GO TO 105
      IF(A(N(8),N(9)).EQ.0.0) GO TO 105
      N110=N19+A(N(8),N(9))
      IF(N110.GE.CORDIS) GO TO 105
      IF(A(N(9),N(10)).EQ.0.0) GO TO 105
      N111=N110+A(N(9),N(10))
      IF(N111.GE.CORDIS) GO TO 105
      IF(A(N(10),1).EQ.0.0) GO TO 105
      DIST=N111+A(N(10),1)
      IF(DIST.GT.CORDIS) GO TO 105
      IF(DIST.LT.CORDIS) GO TO 113
      ISOL=ISOL+1
      GO TO 114
  113 ISOL=1
  114 DO 115 IT=1,M1
  115 ANS(ISOL,IT)=N(IT)
      CORDIS=DIST
      GO TO 105
  103 ICT362=ICT362+1
      ICT403=0
      ICT504=0
      ICT720=0
      ICT120=0
      ICT24=0
      ICT6=0
      ICT=0
      IRESET=0
      GO TO 2
  104 ICT403=ICT403+1
      ICT504=0
      ICT720=0
      ICT120=0
      ICT24=0
      ICT6=0
      ICT=0
      IRESET=13
      GO TO 3
  108 ICT504=ICT504+1
      ICT720=0
      ICT120=0
      ICT24=0
      ICT6=0
      ICT=0
      IRESET=14
      GO TO 5
  109 ICT720=ICT720+1
      ICT120=0
      ICT24=0
      ICT6=0
      ICT=0
      IRESET=15
      GO TO 7
  110 ICT120=ICT120+1
      ICT24=0
      ICT6=0
      ICT=0
      IRESET=16
      GO TO 9
  111 ICT24=ICT24+1
```

```
      ICT6=0
      ICT=0
      IRESET=17
      GO TO 11
  112 ICT6=ICT6+1
      ICT=0
      IRESET=18
      GO TO 13
C************************************ M=12 ***************************
   88 N(11)=1
      DO 89 I11=1,M11
   90 N(11)=N(11)+1
      IF(N(11).EQ.N(10)) GO TO 90
      IF(N(11).EQ.N(9)) GO TO 90
      IF(N(11).EQ.N(8)) GO TO 90
      IF(N(11).EQ.N(7)) GO TO 90
      IF(N(11).EQ.N(6)) GO TO 90
      IF(N(11).EQ.N(5)) GO TO 90
      IF(N(11).EQ.N(4)) GO TO 90
      IF(N(11).EQ.N(3)) GO TO 90
      IF(N(11).EQ.N(2)) GO TO 90
      IF(N(11).EQ.N(1)) GO TO 90
      IF(ICT.NE.6) GO TO 701
      ICT6=ICT6+1
      ICT=0
      IRESET=19
  701 IF(ICT6.NE.4) GO TO 702
      ICT24=ICT24+1
      ICT6=0
      IRESET=18
  702 IF(ICT24.NE.5) GO TO 703
      ICT120=ICT120+1
      ICT24=0
      IRESET=17
  703 IF(ICT120.NE.6) GO TO 704
      ICT720=ICT720+1
      ICT120=0
      IRESET=16
  704 IF(ICT720.NE.7) GO TO 705
      ICT504=ICT504+1
      ICT720=0
      IRESET=15
  705 IF(ICT504.NE.8) GO TO 706
      ICT403=ICT403+1
      ICT504=0
      IRESET=14
  706 IF(ICT403.NE.9) GO TO 707
      ICT362=ICT362+1
      ICT403=0
      IRESET=13
  707 IF(ICT362.NE.10) GO TO 708
      ICT36M=ICT36M+1
      ICT362=0
      IRESET=0
  708 ICT=ICT+1
C     *****CHECK REVERSE*****
      IF(N(11).LT.N(1)) GO TO 89
      DIST=0.0
      IF(IRESET.EQ.20) GO TO 709
      IF(IRESET.EQ.19) GO TO 710
      IF(IRESET.EQ.18) GO TO 711
      IF(IRESET.EQ.17) GO TO 712
      IF(IRESET.EQ.16) GO TO 713
      IF(IRESET.EQ.15) GO TO 714
      IF(IRESET.EQ.14) GO TO 715
      IF(IRESET.EQ.13) GO TO 716
      IRESET=20
      IF(A(1,N(1)).EQ.0.0) GO TO 717
      N12=A(1,N(1))
      IF(N12.GE.CORDIS) GO TO 717
  716 IF(A(N(1),N(2)).EQ.0.0) GO TO 718
      N13=N12+A(N(1),N(2))
      IF(N13.GE.CORDIS) GO TO 718
  715 IF(A(N(2),N(3)).EQ.0.0) GO TO 719
      N14=N13+A(N(2),N(3))
      IF(N14.GE.CORDIS) GO TO 719
  714 IF(A(N(3),N(4)).EQ.0.0) GO TO 720
      N15=N14+A(N(3),N(4))
      IF(N15.GE.CORDIS) GO TO 720
  713 IF(A(N(4),N(5)).EQ.0.0) GO TO 721
```

```
      N16=N15+A(N(4),N(5))
      IF(N16.GE.CORDIS) GO TO 721
  712 IF(A(N(5),N(6)).EQ.0.0) GO TO 722
      N17=N16+A(N(5),N(6))
      IF(N17.GE.CORDIS) GO TO 722
  711 IF(A(N(6),N(7)).EQ.0.0) GO TO 723
      N18=N17+A(N(6),N(7))
      IF(N18.GE.CORDIS) GO TO 723
  710 IF(A(N(7),N(8)).EQ.0.0) GO TO 724
      N19=N18+A(N(7),N(8))
      IF(N19.GE.CORDIS) GO TO 724
  709 IF(A(N(8),N(9)).EQ.0.0) GO TO 89
      N110=N19+A(N(8),N(9))
      IF(N110.GE.CORDIS) GO TO 89
      IF(A(N(9),N(10)).EQ.0.0) GO TO 89
      N111=N110+A(N(9),N(10))
      IF(N111.GE.CORDIS) GO TO 89
      IF(A(N(10),N(11)).EQ.0.0) GO TO 89
      N112=N111+A(N(10),N(11))
      IF(N112.GE.CORDIS) GO TO 89
      IF(A(N(11),1).EQ.0.0) GO TO 89
      DIST=N112+A(N(11),1)
      IF(DIST.GT.CORDIS) GO TO 89
      IF(DIST.LT.CORDIS) GO TO 725
      ISOL=ISOL+1
      GO TO 726
  725 ISOL=1
  726 DO 727 IT=1,M1
  727 ANS(ISOL,IT)=N(IT)
      CORDIS=DIST
      GO TO 89
  717 ICT36M=ICT36M+1
      ICT362=0
      ICT403=0
      ICT504=0
      ICT720=0
      ICT120=0
      ICT24=0
      ICT6=0
      ICT=0
      IRESET=0
      GO TO 2
  718 ICT362=ICT362+1
      ICT403=0
      ICT504=0
      ICT720=0
      ICT120=0
      ICT24=0
      ICT6=0
      ICT=0
      IRESET=13
      GO TO 3
  719 ICT403=ICT403+1
      ICT504=0
      ICT720=0
      ICT120=0
      ICT24=0
      ICT6=0
      ICT=0
      IRESET=14
      GO TO 5
  720 ICT504=ICT504+1
      ICT720=0
      ICT120=0
      ICT24=0
      ICT6=0
      ICT=0
      IRESET=15
      GO TO 7
  721 ICT720=ICT720+1
      ICT120=0
      ICT24=0
      ICT6=0
      ICT=0
      IRESET=16
      GO TO 9
  722 ICT120=ICT120+1
      ICT24=0
      ICT6=0
      ICT=0
```

```
      IRESET=17
      GO TO 11
  723 ICT24=ICT24+1
      ICT6=0
      ICT=0
      IRESET=18
      GO TO 13
  724 ICT6=ICT6+1
      ICT=0                                   555
      IRESET=19
      GO TO 21
   89 CONTINUE
  105 CONTINUE
   24 CONTINUE
   21 CONTINUE
   13 CONTINUE
   11 CONTINUE
    9 CONTINUE
    7 CONTINUE
    5 CONTINUE
    3 CONTINUE
    2 CONTINUE
C***************** THIS SECTION PRINTS THE RESULTS *******************
C     IN THE EXTREME CASE WHERE THE FIRST ROUTE EXAMINED EQUALS THE
C     UPPER BOUND CORDIS THEN THIS NEXT LOOP WILL PRINT AN UNDEFINED
C     ROUTE AS THE FIRST SOLUTION FOLLOWED BY THE CORRECT SOLUTION.
C*********************************************************************
      N1=M1/10
      FMTOVL(38)=DIGIT(N1+1)
      FMTOVL(39)=DIGIT(N2+1)
      IHOLD=ISOL
      DO 32 ISOL=1,IHOLD
   32 WRITE(6,FMT) (ANS(ISOL,IT),IT=1,M1)
      WRITE(6,30) CORDIS
   30 FORMAT('0','THE ROUTE IS',F9.2,'  MILES IN LENGTH')
      WRITE(6,830) UPLIM
  830 FORMAT('0','THE INITIAL UPPER BOUND USED WAS',F7.1,' MILES')
      GO TO 9998
 9999 STOP
      END
      SUBROUTINE PRINT(T,K)
      REAL*8 NUMS(15,15), BLANK/'    NL'/,DIAGON/'    * '/
      REAL T(15,15)
      WRITE(6,4)
    4 FORMAT('-',52X,'BETWEEN CITY DISTANCE MATRIX')
      WRITE(6,10) (J,J=1,15)
   10 FORMAT('0',21X,15I6)
      WRITE(6,11)
   11 FORMAT(22X,90('_'))
      WRITE(6,17)
   17 FORMAT('0')
      DO 6 I=1,K
      DO 6 J=1,K
      IF(I.EQ.J) GO TO 12
      IF(T(I,J).NE.0.0) GO TO 13
      NUMS(I,J)=BLANK
    5 FORMAT(F6.1)
      GO TO 6
   12 NUMS(I,J)=DIAGON
      GO TO 6
   13 CALL CORE(NUMS(I,J),6)
      WRITE(99,5) T(I,J)
    6 CONTINUE
      DO 8 I=1,K
      WRITE(6,9) I,(NUMS(I,J),J=1,K)
    9 FORMAT(' ',16X,I3,1X,'|',15A6)
      WRITE(6,7)
    8 WRITE(6,7)
    7 FORMAT(21X,'|')
      WRITE(6,99)
   99 FORMAT('1')
      RETURN
      END
C*********************************************************************
C     THIS FUNCTION RETURNS A VALUE NUMFAC AND ASSIGNS IT TO ISIZE.
C     THIS VALUE REPRESENTS THE NUMBER OF CIRCULAR PERMUTATIONS (FEA-
C     SIBLE ROUTES) FOR AN M CITY PROBLEM.
C*********************************************************************
      INTEGER FUNCTION NUMFAC(M)
      NUMFAC=2
```

```
      ICOUNT=3
      NN=M-3
      DO 1 K=1,NN
      NUMFAC=NUMFAC*ICOUNT
    1 ICOUNT=ICOUNT+1
      NUMFAC=NUMFAC/2
      RETURN
      END
      BLOCK DATA
      COMMON/FORMAT/FMT
      COMPLEX*16 FMT(4)/'(''0'',''THE SHORTE','ST ROUTE IS'',4X,',''' 1'
     *',M1(''-'',I2)',''-1'')'/
      END
```

THE SOLUTION OF A MILK-TRUCK ROUTING PROBLEM VIA TRAVELING SALESMAN ANALYSIS: AND THE DEVELOPMENT OF AN ALTERNATIVE APPROACH

by

WALTER LYNN TURNER

B. S., Kansas State University, 1974

AN ABSTRACT OF A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF BUSINESS ADMINISTRATION

College of Business Administration

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1976

Various approaches to the traveling salesman problem were considered in the selection of an approach to apply to a real-world, eleven-city milk routing problem. Two approaches were chosen, one using a general purpose 0-1 integer programming algorithm (RIP 30C) and another using a combinatorial approach developed for this thesis (CITDIS).

CITDIS examines all possible routes by generating a subset of them and implicitly considering the remainder. The pattern in the route generation is used to save computations by bounding and by eliminating reverse routes. In addition, the existence of no-links (no connecting road) between cities coupled with the pattern of route generation allows further reduction in computation time.

The integer programming approach to the milk routing problem was found to be unsatisfactory due to computational time limits. CITDIS solved the eleven-city milk routing problem in under ten seconds (on an IBM 370-158 computer).

Implications of resolution of the milk routing problem were discussed. In addition, these implications were generalized to the application of traveling salesman analysis in areas other than routing.