1970

A coordinate oriented algorithm for the traveling salesman problem

Joseph Sidney Greene

Department: Mathematics and Statistics

A COORDINATE ORIENTED ALGORITHM FOR THE TRAVELING SALESMAN PROBLEM

BY

JOSEPH SIDNEY GREENE, SR., 1932

A DISSERTATION

Presented to the Faculty of the Graduate School of the

UNIVERSITY OF MISSOURI - ROLLA

In Partial Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

in

MATHEMATICS

1970

A K Rigler
Advisor

B E Gillett

Troy L. Hicks

Lonny B. Winrich

Harold Dean Keith

ABSTRACT

The traveling salesman problem may be stated as follows: "A salesman is required to visit each of n given cities once and only once, starting from any city and returning to the original place of departure. What route should be chosen in order to minimize the total distance traveled?"

A new algorithm is developed which gives a good approximation to the solution for a large number of cities using reasonable computer time and which will converge to the exact solution if allowed to continue.

This algorithm is a branch and bound technique which utilizes the distance between cities in its bounding procedure. The bookkeeping scheme for the algorithm is such that only the partial solution along with those routes currently being checked need be retained in memory. The branching technique requires that only one row of the distance matrix be in memory at any time.

The algorithm is demonstrated using a four-city problem and a formal statement is given. Computational results from computer implementation of the algorithm are given, including three realistic problems from the printed circuit industry.

ACKNOWLEDGEMENTS

First, the author wishes to express his sincere appreciation to Dr. A. K. Rigler of the Computer Science Department and to Dr. Billy E. Gillett, Chairman of the Computer Science Department for their help and guidance through the preparation of this dissertation.

The author also wishes to thank Professor Ralph E. Lee and the staff of the University of Missouri-Rolla Computer Center for their ever present help in testing the algorithm developed in this dissertation as well as other algorithms. Two former students, Steven N. Nau and Paula Graves also saved the author many hours of waiting and did much of the tedious testing.

Also, the author is indebted to the National Science Foundation for their financial support during the course of his study. In addition the author wishes to thank his associates of the Advanced Circuitry Division of Litton Precision Products for their cooperation and technical support. To Mrs. Charlotte Beazley goes a special thanks for her exceptional skill and patience in typing this manuscript.

Finally to my wife Joanne, son Joey, and daughter Julie go a very special expression of gratitude for their encouragement, patience, understanding, and sacrifices during the years of study.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure Page

Plates

LIST OF TABLES

I. INTRODUCTION

A. Statement of the Problem

The traveling salesman problem is easy to state: A salesman starting in one city must visit each of n-1 other cities once and only once and return to the originating city. What should be the order of the visits if the salesman wishes to minimize the distance traveled? Considering distance as symmetric we see that there are (n-1)!/2 possible tours, one (or more) of which must give the minimum distance.

B. History

The origin of the traveling salesman problem is attributed by Dantzig (1) to a seminar talk given by Hassler Whitney at Princeton in 1934.* In 1937 Merrill Flood, who also credits Whitney with the origin of the problem, applied the problem to school bus routing (2). The problem is closely related to problems considered by Hamilton in which he tried to determine the number of different tours possible over a specified network (3). In their survey paper (4), Bellmore and Nemhauser credit Flood with early stimulation of research in the problem. The Rand Corporation offered a prize for any significant theorem relating to the problem, but in 1956 Flood reports that no award had been made. In 1962 a soap company offered

* All numbers (a) refer to the bibliography while the numbers (a,b) refer to equations.

prizes of up to $10,000.00 for identifying the best route in a particular 33-city problem (5). This gave national recognition to the problem and motivated further research in the area.

The survey of the problem by Bellmore and Nemhauser lists 10 theorems related to the problem. These same authors list several methods of solution to the problem and give the computational experience when available. Some of these methods will be explored in greater detail below.

C. A New Application

The circuit board industry furnished motivation for making further study of the traveling salesman problem. Circuit boards (many of which are used in computers) usually have numerous holes, sometimes more than one thousand. These holes are drilled by a numerically controlled printed circuit board drilling machine. In this context the holes in the circuit board correspond to cities, and the movement of the drill head above the boards to travel or distance. We might point out that movement can be made only in the horizontal or vertical direction. This would correspond to travel by city blocks in a routing problem, for example, a taxicab routing problem. A fringe benefit of this application is that data is readily available on paper tape. That is, the data is given in the form of points on a rectangular coordinate system and the order of the points represents a feasible solution. This solution is obtained manually by a well trained technician and may be nearly optimal. This is an instance where one generation of computers is used to help control cost on computers of future generations.

In the industry several circuit boards are placed on a panel for the manufacturing process, (see Plate 1). In order to find the shortest route through a board we need a shortest route algorithm. To find the shortest path connecting all boards a traveling salesman algorithm is needed.

Thus it is the purpose of this dissertation to develop an algorithm suitable for application in the circuit board industry. This dictated the following requirements:

1. That all constraints be generated by the program.

2. That the algorithm can be stopped at any time giving a "current best solution."

3. That the algorithm make rapid progress toward convergence early in the computational stages.

4. That the algorithm converge to the exact solution if computer time is available and if circumstances warrant the expenditure.

5. That the algorithm be readily usable either as a traveling salesman or shortest route algorithm.

D. An Overview

In Chapter II a review of the literature including several formulations of the problem is given. The emphasis is placed on those procedures which satisfy at least one criterion stated above.

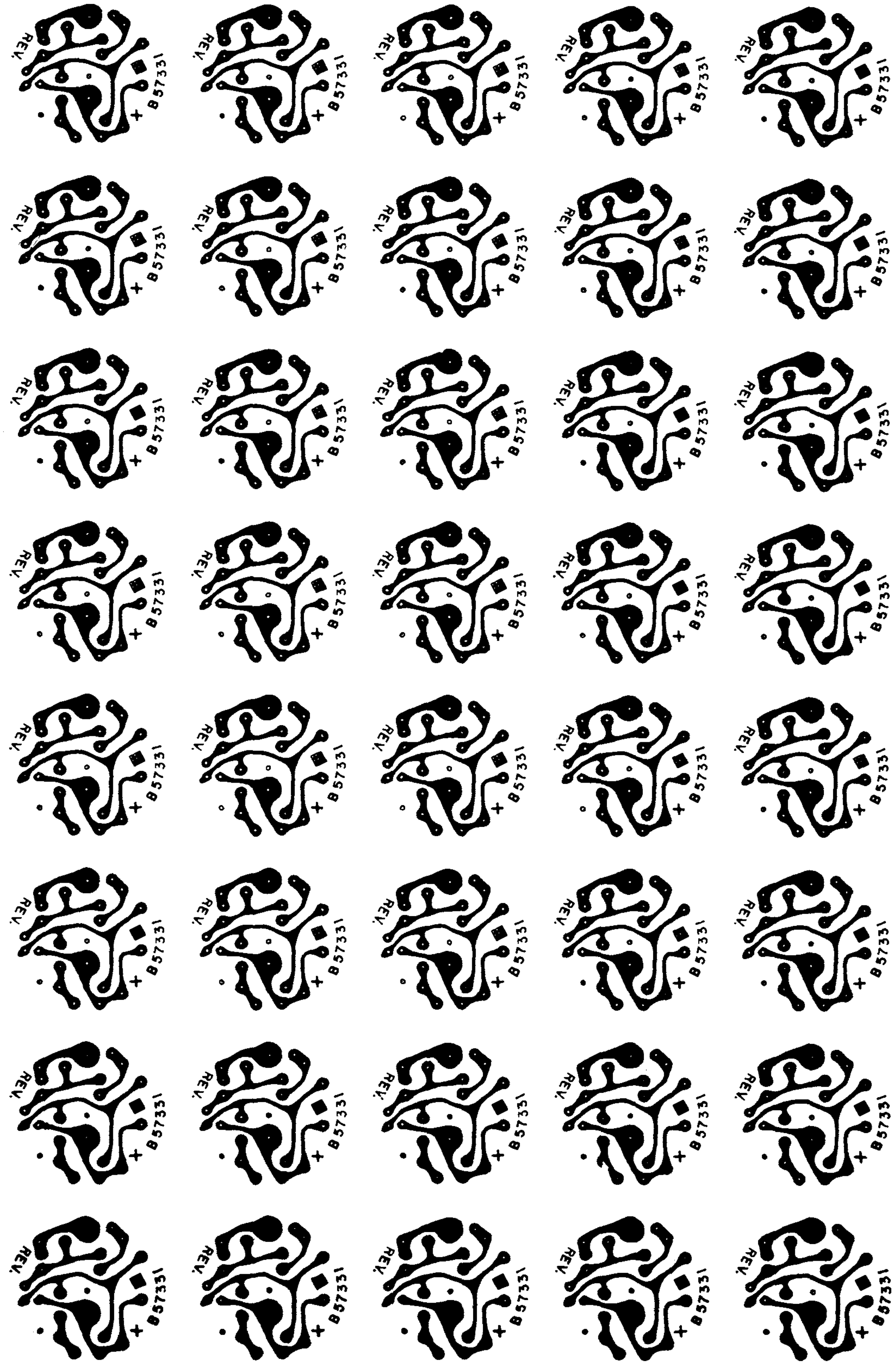
Chapter III contains the development of a new coordinate oriented traveling salesman algorithm. Two new theorems are presented which play a major role in the development of the algorithm.

A discussion of how the algorithm differs from those presented in Chapter II is given in Chapter IV. The chapter concludes with an example to illustrate how the algorithm is applied.

Computational results involving the algorithm are presented in Chapter V. In particular, five problems were constructed for testing purposes. These problems were constructed so that the points fell on a rectangle or circle so the minimums were readily available. Also sixteen problems were randomly generated for comparison purposes. The chapter concludes with three live problems from the printed circuit industry and economic considerations indicating when to use the algorithm.

Chapter VI contains the conclusions and suggestions for further study.

PLATE I



II. REVIEW OF THE LITERATURE

All available literature concerning the traveling salesman problem was examined; however, emphasis was placed on those methods which satisfied at least one of the criteria as stated in the introduction.

After stating several formulations of the problem some of the more successful known algorithms are presented.

A. Formulations of the Problem

The problem can be formulated in terms of finding a permutation of the first n natural numbers such as;

$P = (i_1, i_2, \ldots, i_n)$ which minimizes the quantity

$\sum_{k=1}^{n-1} d_{i_k, i_{k+1}} + d_{i_n, i_1}$ where the d_{pq} designate real numbers corresponding to the distance between city p and city q. In this context the problem may or may not be symmetric, that is d_{pq} may or may not equal d_{qp}.

The distances between each city can be written in matrix form by defining $d_{ii} = 0$ for all $i = 1, \ldots, n$, and $D = (d_{ij})$. Then one can describe the problem in terms of the distance matrix D as follows:

Determine x_{ij} which minimizes

$$F = \sum_{ij}\sum d_{ij} x_{ij} \text{ subject to}$$

$$x_{ii} = 0 \tag{1}$$

$$x_{ij} = 0,1 \tag{2}$$

$$\sum_j x_{ij} = \sum_i x_{ij} = 1, \text{ and for any subset of the} \tag{3}$$

first n natural numbers $P = \{i_1, i_2, \ldots, i_r\}$ we have

$$x_{i_1i_2} + x_{i_2i_3} + x_{i_3i_4} + \cdots + x_{i_ri_1}$$

$$\begin{aligned} &< r \text{ for } r < n \\ &\leq n \text{ for } r = n. \end{aligned} \tag{4}$$

If $x_{ij} = 1$ then one travels from city i to city j and if $x_{ij} = 0$ one does not. The constraints (II,3) guarantee that each city is visited once and only once while (II,4) eliminates subtours. Recall that each city is to be visited before returning to the originating city. A subtour is also called a cycle. It is this last constraint which distinguishes the traveling salesman problem from the assignment problem. Note that the latter formulation is a 0,1 integer programming problem also.

The formulation of Tucker (6) was selected as the most promising because the number of constraints necessary is less than the number required for the above formulations. In addition, it is possible to generate the constraints that prevent cycling as they are needed. If d_{ij} again represents the distance from city i to city j Tucker's formulation can be stated thus:

minimize $$\sum\sum_{1 \leq i \neq j \leq n} d_{ij}x_{ij}$$

subject to

$$x_{ij} = 0,1$$

$$\sum_{\substack{i=1\\ i\neq j}}^{n} x_{ij} = 1 \quad (j = 1, \ldots, n) \tag{5}$$

$$\sum_{\substack{j=1\\ j\neq i}}^{n} x_{ij} = 1 \quad (i = 1, \ldots, n) \tag{6}$$

$$u_i - u_j + nx_{ij} < n-1 \ (1 \leq i \neq j \leq n). \tag{7}$$

Again the equations represented by (II,5), and (II,6) guarantee that each city is visited once and only once. The equations represented by (II,7) is an unusual way to prevent cycling. Here the u's represent arbitrary real numbers whose sole function is to prevent cycling. The proof that this is true is also given in the above cited article (6).

This formulation also requires fewer variables than the above formulation. However the article states that a four-city problem required 13 constraints and 9 variables. In general an n-city problem requires $n^2 + n$ inequalities and n^2 variables. These can be reduced somewhat by a judicious choice of the slack variables.

From the point of view of graph theory, we may consider the n cities as vertices of a nondirected complete graph, and the entries d_{ij} of the distance matrix real numbers assigned to links x_{ij} connecting city i to city j. A permutation $P = (i_1, i_2, \ldots, i_n)$ representing a tour may be considered as a collection of n links $x_{i_1 i_2}$, $x_{i_2 i_3}$, ..., $x_{i_n i_1}$ forming a Hamiltonian circuit, and the quantity $C = d_{i_1 i_2} + d_{i_2 i_3} + \ldots + d_{i_n i_1}$ the cost associated with the tour.

Finally there is the matrix representation of the problem. By this we mean that a distance matrix D is defined as in the integer programming formulation. However, this is the entire formulation in that the necessary constraints are implicit in the method of transforming the distance matrix. That is, there are algorithms that make only transformations on the distance matrix. This will be called the matrix formulation.

B. Gomory's Cutting Plane Method

Gomory's method (7) uses the integer programming formulation of the problem. First the linear programming problem

$$\max F = \sum_{0 \leq i \neq j \leq n} \sum d_{ij} x_{ij} \qquad (8)$$

subject to

$$\sum_{\substack{i=0 \\ i \neq j}}^{n} x_{ij} = 1 \; (j = 1, \ldots, n) \qquad (9)$$

$$\sum_{\substack{j=0 \\ i \neq j}}^{n} x_{ij} = 1 \; (i = 1, \ldots, n) \qquad (10)$$

$$u_i - u_j + nx_{ij} < n-1 \quad (1\leq i\neq j\leq n) \qquad (11)$$

is formulated.

Gomory's algorithm is best suited for a maximization problem, hence the appropriate changes of signs are necessary in equation (II,8). The number of cities is represented by n.

Now the simplex algorithm is used to solve this problem. If all $x_{ij} = 0$, or 1 the algorithm terminates. Otherwise there is a non-integer x_{ij} in the solution. Thus in the final simplex tableau we have a row such as

$$ox_{i1} + \ldots + x_{ij} + \ldots + ox_{in} + y_1x_{i,n+1} + \ldots + y_kx_{i,n+k} = t.$$

Here the $x_{i,n+1}$ to $x_{i,n+k}$ represent slack and surplus variables. Hence we read

$$x_{ij} = t \text{ and } x_{im} = 0 \text{ for } m > n$$

where t is not an integer. A new constraint is now generated in the form

$$y'_1x_{in} + \ldots + y'_kx_{i,n+k} \geq t' \qquad (12)$$

where y'_m and t' are the smallest nonnegative numbers congruent to y_m and t respectively.

This new constraint is annexed to the constraints of the last tableau and this new problem is solved using the dual simplex algorithm. The dual algorithm is used because the simplex terminating criterion has been satisfied. Also, when the new constraint is added the new tableau is in the desired form for the dual algorithm. The process continues until all variables are integers.

In his paper Gomory shows that equations of the form (II,12) form a necessary condition that will prevail when all variables have their optimum integer values. In order to accelerate convergence Gomory suggests that the variable with the largest fractional part be used to generate the new constraint.

A computer code for this algorithm is available in both FORTRAN and ALGOL (8).

This algorithm was discarded because the number of constraints became prohibitive even for small (five-city) problems. This is due in part to the fact that the algorithm utilizes both the simplex and dual simplex techniques. Because the dual is used each equation (II,3) must be replaced by two inequalities.

C. Land and Doig's Branch and Bound

Land and Doig's algorithm (9) also utilizes the integer programming formulation of the problem. First the relaxed problem (II,8), (II,9), (II,10), and (II,11) is solved by the simplex technique. This solution gives a bound for an all integer solution. If this solution contains a non-integer x_{ij} say

$$x_{ij} = t,$$

two new problems are generated. The first is obtained by annexing to the original problem given by (II,8), (II,9), (II,10), and (II,11) the additional constraint

$$x_{ij} = [t].^*$$

This is called branch 1. The second problem is obtained by annexing to the original problem the additional constraint

$$x_{ij} = [t] + 1.$$

This is called branch 2.

These 2 problems are solved using the simplex algorithm. The resulting objective function values constitute bounds on all further constrained problems in their respective classes. If the solution with the largest value of objective function has a non-integer value another branch is made. That is 2 additional problems are generated. Suppose branch 1 has the maximum objective function and

$$x_{kl} = s$$

where s is not an integer. To the original equations for branch 1 we annex

$$x_{kl} = [s]$$

to obtain branch 3. To obtain branch 4

$$x_{kl} = [s] + 1$$

is annexed to branch 1. Again these two problems are solved using the simplex technique given additional bounds. We now have three terminal branches, 2, 3, and 4. Of these, the one with maximum

* [a] indicates the entier function.

objective function value is chosen. If this branch contains a non-integer value two new branches are constructed as above. This process is continued until the maximum terminal branch has all integer values.

This algorithm worked well for three and four-city problems. However, it was found that auxiliary storage* was necessary for larger problems. This increased convergence time significantly. For a five-city problem computer time was excessive. The code we used is found in McMillan (10).

D. The Algorithm of Balas

For Balas' algorithm (11), (12) the integer programming formulation is also used. The problem is written for the algorithm as

minimize

$$F = \sum_{i=1 \neq j=1}^{n} \sum^{n} d_{ij} \; x_{ij}$$

subject to

$$-1 + \sum_{\substack{i=1 \\ i \neq j}}^{n} x_{ij} = 0 \; (j = 1, \ldots, n)$$

$$-1 + \sum_{j=1}^{n} x_{ij} = 0 \; (i = 1, \ldots, n)$$

$$n-1 - nx_{ij} + u_j - u_i \geq 0 \; (i\_i \neq \leq n).$$

* Auxiliary storage is defined as magnetic tape or magnetic disk storage.

That is, each constraint is written in the form of ≥ 0. The integer constraints are not needed as this is a 0-1 algorithm.

The algorithm begins with the infeasible solution (0, ..., 0). The constraints are evaluated for this solution and a measure of infeasibleness is made. This measure is the sum of the amount less than zero over all violated constraints. This is called the "test measure."

Next the variable which reduces the test measure by the largest amount is set to 1. This partial solution is also checked for infeasibility. If the solution is infeasible the above process is repeated and a new variable is set to 1. This process continues until a feasible solution is obtained.

Now no other variable would be raised to 1 as this would increase the value of the objective function. A backtracking scheme begins at this point.

Consider a solution vector of the form

$$(1, 0, 0, 1, 1, \ldots, 0, 1)$$

which gives a value for the objective function, say z. The variable which was last entered into the solution is set "free." That is, its value is set to 0, but for testing and bookkeeping purposes it is given the value -1 (underlined). Now the test measure for this partial solution is made and the free variable with maximum increase becomes a candidate for entry into the solution.

If the value of the partial solution, that is $\Sigma\Sigma\ d_{ij}\ x_{ij}$ where

x_{ij} = 1, plus the distance d_{ij} associated with the new candidate is less than z the candidate is entered into the solution. Z is set to the value of the objective function for this solution and backtracking continues. Backtracking continues by underlining the last variables which was set to 1 and which has not yet been underlined.

When all elements in the original solution have been underlined the algorithm terminates as all solutions have been enumerated either explicitly or implicitly.

The algorithm was revised somewhat to fit the problem at hand. For example, the first feasible solution was generated from the order in which the points were read. The code used for testing can be found in McMillan (10).

This algorithm worked well for three and four city problems, but again computer time was excessive for larger problems. For example, a five-city problem took five minutes on the IBM-360/50. Revision of the code and algorithm led to reductions of approximately three minutes for a five-city problem. Balas' algorithm was reluctantly abandonded because it satisfied all requirements, stated in the introduction, except reasonable convergence time.

E. Little's Branch and Bound

One of the more successful methods for exact solutions of the traveling salesman problem is that due to Little, et al, (5).

Little's procedure uses the matrix representation of the problem. The algorithm may be described in general as a method of splitting the tour solution space into disjoint subspaces with a concomitant increase in the lower bound on the solution. The subspace with the smallest lower bound is used as a basis for further splitting. This process is terminated when a subspace is found which contains only a tour solution and whose lower bound distance is less than or equal to that of every other derived subspace.

A theorem found in (3) and (13) is used to establish the initial lower bound on all tours. Constants are subtracted from rows and columns of the cost matrix until there is at least one zero and no negative elements in every row and column. By the theorem, the sum of these constants is a lower bound on all tours. Next, the splitting procedure begins.

At every stage, the zeros of the matrix are examined. The splitting is accomplished by means of a single variable x_{ij} whose cost entry is zero. The subspace is split into two subspaces representing those tours containing x_{ij} (denoted by x_{ij} or (i,j)) and those not containing x_{ij} (denoted by $\overline{x}_{ij}$ or $(\overline{i},\overline{j})$).

1. If x_{ij} is not on the optimal tour then the minimum cost incurred must be

$$t_{ij} = \underset{k \neq j}{\text{Min}}\, A_{ik} + \underset{m \neq i}{\text{Min}}\, A_{mj}.$$

The x_{ij} whose value t_{ij} is maximal is chosen as a basis for further splitting.

2. If x_{ij} is on the optimal tour, cross out row i and column j, prohibit all subtours containing x_{ij} and those variables already committed and see if the matrix may be further reduced so that at least one zero appears in every row and column (not crossed out).

The new lower bounds are computed as follows:

$$W_1(\bar{x}_{ij}) = W + t_{ij}$$

$$W_2(x_{ij}) = W + R$$

where W is the lower bound on the subspace before splitting and R is the sum of reducing constants.

Subtours are prohibited by setting

$$A_{k,m} = \infty$$

where (k,m) is the closure of the longest subtour involving x_{ij} and those variables already committed. Only the zeros of the matrix need be examined since for any other element $t_{ij} = 0$.

When n-2 variables have been committed to any subspace, this subspace represents a tour. If its lower bound is minimal it is the optimal tour. If not, go to the subspace of minimum lower bound and continue the procedure. Once a tour has been established, any subspace whose lower bound exceeds the tour cost may be disregarded. This algorithm was not tested since comprehensive tests were made by Shapiro (14) and Sweeny (15). Shapiro reports that for a 25-city problem the algorithm generated in excess of 3,000 subspaces without converging to the minimum. Both Shapiro and Sweeny report difficulty in solving symmetric problems.

F. Local Optimal Algorithms

The most successful locally optimal solution previously published is due to Lin (16). This algorithm uses the graph theory formulation of the problem. Lin's algorithm begins with a random tour and the procedure can be summarized in the following definition: A tour is said to be λ-optimal (λ-opt) if it is impossible to obtain a shorter tour by replacing any λ of its links by any other λ links.

Croes (17) applied a simple transformation called "inversions" to transform a trial solution into another with shorter distance, iterating until no further inversions are possible. Croes showed this eliminated routes that cross, which are not optimal (3).

Lin shows that for $\lambda=3$ his algorithm gives inversion-free tours with average distance considerably less than that given by Croes' inversion method.

However, for $\lambda>3$ Lin reports that convergence time for the algorithm was prohibitive and accuracy was not improved significantly.

Lin's algorithm is used extensively in Chapter IV for comparison purposes. The code for the algorithm was furnished by Miller (18).

III. THE ALGORITHM

A. The Problem

Given a set of n points

$$P_1, P_2, \ldots, P_n$$

in a plane it is required to generate a traveling salesman problem and obtain a solution for it. The algorithm used should have the properties stated in the introduction. In this setting the points correspond to cities and distance between them can be computed by any desired formula.

B. Definitions and Notation

To facilitate the description of the algorithm we make the following definitions.

1. n as the number of cities.

2. Let μ and ν be the vectors of ordinates and abscissas of the cities.

3. Let $D = (d_{ij})$ be the matrix of distances from city i to city j. d_{ii} is not defined for all i. The problems used from the printed circuit industry dictates that

$$d_{ij} = |\mu_i - \mu_j| + |\nu_i - \nu_j| \text{ where } i \neq j.$$

4. ψ the solution vector currently under consideration,

containing row and column numbers in order of consideration.

5. iz the distance associated with the partial solution vector currently under consideration.

6. ω the vector of the incumbent best feasible route.

7. z the distance of the incumbent route.

8. α the vector designating whether or not row i has been considered in the backtracking scheme. If α_i=0 then row i has not been reached in the backtracking scheme. If α_i=1 then i is the highest row under consideration. α_i=-1 designates that row i has been tested in the backtrack scheme.

9. β the vector whose i elements designates the number of elements which have been underlined in row i.

10. k the row currently under consideration.

11. γ the vector of minimum row values.

12. ns the number of underlines required in the current iteration.

13. Index the number of values currently in ψ which also includes the underlined elements.

14. "Free" columns are those columns which are eligible for

entry into the solution.

In the discussion α_{i}, β_{i}, ... represent the ith elements in the vectors α, β, ..., .

C. Statement of Theorems

As the theorems play a major role in the algorithm they are presented here for reference purposes. The proofs will follow in Section F of this chapter.

Theorem 1. For $\beta_k = 0$, or 1, the row search can begin in column $k + 2$. Furthermore if a column is chosen greater than $k + 1$ the partial solution contains no cycle.

Theorem 2. The algorithm enumerates all solutions.

D. Description of the Algorithm

The algorithm begins by generating the distance matrix D as given above. A first feasible solution is generated by the order in which the points are given. This solution is placed along the upper main diagonal of D. That is, the first solution is: city 1 (point 1) to city 2; city 2 to city 3; ...; city n to city 1. This is recorded as a vector $\omega = \{1,2,2,3,\ldots,n-1,n,n,1\}$. The distance, z, generated by this solution is recorded with the solution ω. Next a vector γ of row minimums is generated. Thus the ith element of γ is the minimum of row i in D.

A vector ψ which is used to record the partial solutions is set equal to ω. Two other vectors α and β are set equal to the zero vector and are used to aid in the bookkeeping as explained below.

At this point a backtracking scheme begins by deleting the pairs (n,1), (n-1,n) from ψ and the pair (n-2,n-1) is replaced by (-n+2,-n+1). The latter operation is called underlining. Now k, the row currently under consideration, is set equal to n-2, α_k is set to 1, and α_i to -1 for $i > k$. Also β_k is set to 1. The values of α indicate whether a particular row has been tested, is the row currently being tested, or is below the row which is currently being tested for solution improvement. The value β_i, indicates the number of elements in row i which has been tested for solution improvement.

Row k is now searched for the "free" column with the minimum value. A free column is a column which has not been assigned in the partial solution or has not been underlined. The value of this minimum is added to iz, the value dictated by the current partial solution. This sum is added to the sum of γ_i for $i > k$. If this sum is less than z, the value of the incumbent solution, and the assignment does not produce a cycle, k is incremented by 1 and the search begins again in row k + 1. However, if the sum is greater than the value of the incumbent solution, k is decreased by 1 and a backtracking process begins.

When backtracking the value of β_k is examined to see if the required number of tests (underlines) have been made on row k. One underline is required for each row on the first iteration, two on the second, and so forth. If the correct number of underlines have been made backtracking continues. If not, the search for a minimum begins as explained in the preceding paragraph.

If the value of k becomes n, an improved solution has been found which replaces the incumbent solution. When the value of k becomes 0, the underlining requirement has been met for every row and the iteration is complete.

To begin the next iteration the vectors μ and ν are reordered to coincide with the order of the incumbent solution. Then the matrix D is again generated with the current solution along the upper main diagonal. For this iteration an additional element in each row must be tested (underlined) for solution improvement before backtracking. This process continues until k = n-1 or until otherwise terminated.

Theorem 2 shows that if n-1 iterations are made the exact solution has been found.

A new matrix D is generated at each iteration in order to utilize Theorem 1. In certain cases Theorem 1 reduces both the number of elements which must be tested for minimum values and also eliminates testing for cycles. These cases are dictated by the values of α.

The general procedure for the algorithm and a flow diagram follow.

E. The Algorithm

Step 0.

Read coordinates of points as given by μ and ν.

Step 1. Initalize.

Generate D from μ and ν. The first value of ω is given by the order in which the points are given. That is

$$\omega = (1\ 2\ 2\ 3\ 3\ 4\ \ldots .\ n-1\ n\ n\ 1).$$

Index is set to 2 times n. ψ is set to ω and iz and z are computed from these routes. The vector of row minimums γ is computed from D. Set k = n-2 and $\alpha_i = \beta_i = 0$ for all i.

Then go to Step 2.

Step 2. Backtrack.

Test to see if the required number of underlines for row k has been made. If so decrease k by 1 and test again. (If the criteria has been satisfied for all rows go to Step 5.) If not, test α_k to determine the branching status of row k. If α_k is positive, then k is the highest row which has been considered in the backtrack scheme. In this case go to Step 2a. If α_k is zero, row k has not yet been used in the backtrack scheme. For $\alpha_k = 0$ go to Step 2b. If α_k is less than zero, row k is below the topmost row which has been considered in the backtrack scheme. If this occurs go to Step 2c.

Step 2a.

Drop all entries in ψ with row value greater than k. Set Index = $2k + 2\beta_k$. Set $\beta_i = 0$ for all i greater than k. Underline the two rightmost entries in ψ and increase β_k by 1 and go to Step 3.

Step 2b.

Set $\alpha_i = -1$ and $\beta_i = 0$ for all i greater than k. Drop all entries in ψ with row value greater than k. Set Index = 2k. Underline the rightmost two entries in ψ. Increase β_k by 1 and go to Step 3.

Step 2c.

Drop all entries in ψ with row value greater than k. Set

$$\text{Index} = 2\sum_{i=1}^{k}\beta_i + 2k$$

Set $\beta_i = 0$ for all i greater than k. Underline the two rightmost elements in ψ and increase β_k by 1. Go to Step 3.

Step 3. Forward Step.

Compute iz for the partial solution. Add to iz the sum of α_i for i greater than k. Search the free columns in row k for the minimum distance. The search will begin at column $2k + 2$ if $\beta_k = 1$ or 0, (see Theorem 1). Otherwise the search will begin in column 1. Augment ψ by the coordinates of this minimum. Add this distance to iz and if iz is less than z increase k by one and go to Step 4; otherwise decrease k by 1 and go to Step 2. Index is used in all calculations which involve ψ.

Step 4.

Test ψ for cycles. If there is no cycle and k is not greater than n, go to Step 3. If k equals n replace ω by ψ and z by iz, set k to n-2 and go to Step 2. If ψ cycles underline the two rightmost entries in ψ, increase β_k by 1 and go to Step 2.

Step 5.

Write the current solution ω and z with the value of ns. If ns is less than n-2 replace ns by ns + 1 and go to Step 6, otherwise terminate.

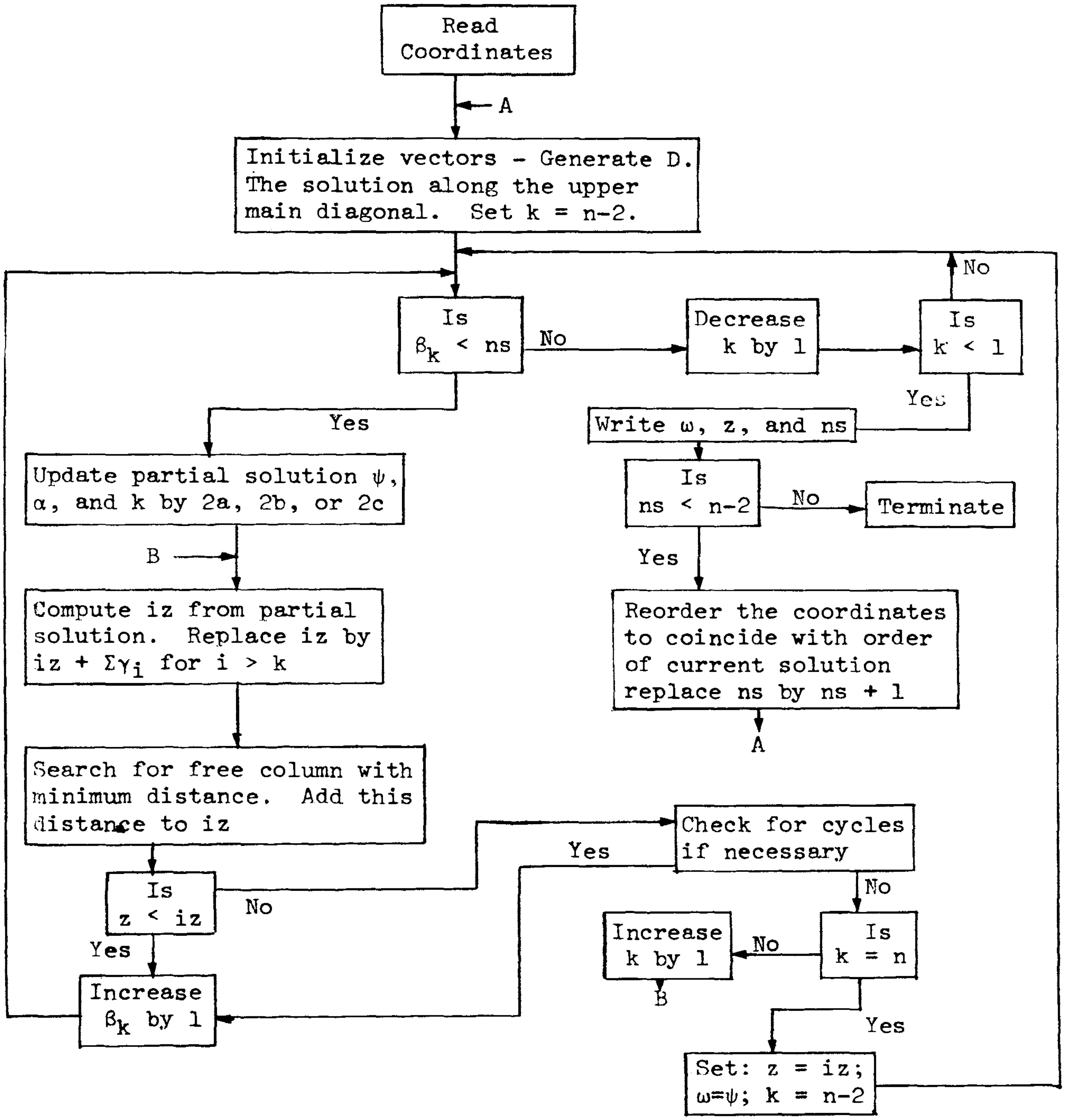
Step 6.

Re-order the coordinates of the vectors μ and ν to coincide with the order of the "current best solution." Go to Step 1.

A simplified flow diagram is given in Figure 1.

Figure 1

Flow Diagram


Read Coordinates
A
Initialize vectors - Generate D. The solution along the upper main diagonal. Set k = n-2.
Is β_k < ns
No
Decrease k by 1
Is k < 1
No
Yes
Write ω, z, and ns
Is ns < n-2
No
Terminate
Yes
Reorder the coordinates to coincide with order of current solution replace ns by ns + 1
A
Yes
Update partial solution ψ, α, and k by 2a, 2b, or 2c
B
Compute iz from partial solution. Replace iz by iz + $\Sigma\gamma_i$ for i > k
Search for free column with minimum distance. Add this distance to iz
Is z < iz
No
Yes
Increase β_k by 1
Check for cycles if necessary
Yes
No
Is k = n
No
Increase k by 1
B
Yes
Set: z = iz; ω=ψ; k = n-2

F. Proofs of Theorems

Theorem 1. For $\beta_k = 0$, or 1, the row search can begin in column $k + 2$. Furthermore if a column is chosen greater than $k + 1$ the partial solution contains no cycle.

Proof:

For $\beta_k = 0$ we have the current partial solution:

$$\psi = \{(1,2), (2,3), \ldots, (k-1,k), (-k, -k-1)\}.$$

Now if we choose the element (k,m) for entry into the solution where $m < k$ then ψ is in the form

$$\psi = \{(1,2), (2,3), \ldots, (m,m+1), \ldots, (k-1,k), (-k,-k-1), (k,m)\}$$

thus the cycle is

$$\{(m,m+1), (m+1,m+2), \ldots, (k-1,k), (k,m)\}.$$

Note that (k,k) is not defined and $(k,k+1)$ is underlined; therefore our search can begin at column $k+2$. For $\beta_k = 1$ the proof is also valid. The only difference being that Ψ contains additional underlined elements.

To show that the new partial solution does not cycle it suffices to consider the partial solution

$$\psi = \{(1,2), (2,3), \ldots, (k-1,k), (-k,-k-1), (k,m)\}$$

where $m > k + 1$. Since m is also greater than every first element in the ordered pairs no cycle is possible. End of proof.

The power of Theorem 1 used in the backtracking scheme lies in two areas:

1. The search can begin to the right of the diagonal and

2. The partial solution obtained in this manner need not be checked for cycling.

It is Theorem 1 that makes this a coordinate oriented algorithm. For in order to use Theorem 1, the initial solution must appear on the upper off diagonal of the distance matrix. Thus after each iteration a new distance matrix must be generated with the initial solution located in this position.

Theorem 2. The algorithm enumerates all solutions.

Consider a new matrix $X = (x_{ij})$ with values 1 or 0, depending on whether d_{ij} is in the current solution or not. That is, $x_{ij} = 1$ means we go from city-i to city-j in the current solution and $x_{ij} = 0$ means we do not. Certainly there are only n^2 elements in X. Now for the solution ω with distance z found in the $ns = n-2$ iteration, X contains n elements with value 1 while all other elements have the value 0. If any of the $x_{ij} = 0$ are set to 1 we have an infeasible solution which also has distance greater than z. Thus no more than n of the n^2 values are 1 and the enumeration can begin by setting these 1's equal to 0.

Now consider the matrix X as a vector δ of 0 and 1's. Here $\delta_{(i-1)n+j}=x_{ij}$. Our backtracking scheme sets the rightmost 1 in δ equal to 0 and considers all other values of 0 for entry into the solution. Next the two rightmost 1's are set to 0 and all values of 0 are considered for entry into the solution except the last one set to 0. This process is continued until all 1's have been set to 0 at which time all solutions have been enumerated. That is, for each i in the original solution, the solutions have been enumerated for which $\delta_i = 1$ and those for which $\delta_i = 0$ and this is all of them. End of proof.

IV. DISCUSSION

A. Preliminaries

Before giving an example illustrating the algorithm the similarities and differences of the algorithm and those presented in Chapter II are noted. First, as with Little, a matrix D of distances is used where the main diagonal elements are not defined. The matrix is not reduced or changed in any way; hence, the bound technique is different from that employed by Little. Also the scheme works with the D matrix systematically from the last row to the first, thus reducing the bookkeeping. That is, the algorithm systematically begins the improvement technique in the last row of the matrix D. Each row is tested in order, from last to first, for route improvement.

The backtracking and enumeration scheme resembles that used by Balas. In particular the bookkeeping scheme parallels that employed by Balas in that a vector of partial solutions is used and a system of underlining those columns checked. However, differing from Balas, the partial solution is augmented by the "free" column with minimum distance which does not produce a cycle and does not violate the bound criteria.

The bound technique is an improvement of Balas' in that to the distance dictated by the current partial solution the sum of the row minimums for the rows below the current row is added.

This distance is tested against the distance of the "current best solution." The bound which Balas uses is the distance given by the partial solution. Recall also that Balas' algorithm uses the integer programming formulation of the problem and uses a branch and bound technique to find the values of 0 and 1 which give the minimum solution. The algorithm of Chapter III is a branch and bound technique based on the matrix representation of the problem.

Another way of viewing the formulation is to note that Balas uses the matrix D as a vector for his cost function. In addition to this vector Balas' formulation also requires a matrix of constraints. Some of these constraints guarantee that each city is visited once and only once while the others eliminate cycling.

The underlining technique requires that one or more elements in the current row be completely tested for solution improvement before backtracking continues. This gives the algorithm the capability of giving either an exact solution or sub-optimal solution.

This idea is similar to that employed in Lin's λ-opt technique. However, there is no similarity between the two algorithms. As Lin uses 3-opt to get a good solution fast, the algorithm tests (underlines) one element in each row before backtracking to obtain a good fast approximation. See Tables I and III in Chapter V for a comparison of these two methods. This improved solution also gives a better bound with which to use the next iteration where it is required that two elements be underlined before backtracking. At any time the "current best solution" is in memory along with the number of underlines currently required.

B. An Example

The following is the matrix representation of a four-city problem. The problem was generated from the following four points located in the first quadrant: (0,0), (10,5), (10,0), (3,5).

	15	10	8
15		5	7
10	5		12
8	7	12	

The first solution was generated from the order of the points. This solution

$$\omega = \{(1,2),\ (2,3),\ (3,4),\ (4,1)\}$$

becomes the "current best solution" and the first working solution. The value of the current best solution 40, becomes the bound for the backtracking operation. Next the vector of row minimums is computed giving

$$\gamma = (8,\ 5,\ 5,\ 7).$$

These values are used along with the value of the partial solution when enumerating solutions.

The first solution is stored in the working vector

$$\psi = \{(1,2),\ (2,3),\ (3,4),\ (4,1)\}$$

and backtracking begins. The last two row assignments are set free and the assignment for the second row is underlined. This is recorded thus:

$$\psi = \{(1,2), (-2,-3)\},$$

iz = 15, and min = 12. The value of the partial solution is iz and min is the sum of the row minimums below the pivotal row.

Now the forward search begins at row two. The minimum free column to the right of the diagonal is chosen as a candidate for entry into the partial solution. A choice to the left of the diagonal would produce a cycle, (see Theorem 1). In this case (2,4) with distance 7 is the candidate. Now the value of the "current best solution" is 40 and

$$15 + 12 + 7 < 40;$$

hence, the partial solution is augmented to

$$\psi = \{(1,2), (-2,-3), (2,4)\},$$

iz = 22, and min = 7. The search now begins in row 3.

In this case the candidate can come from any free column which does not produce a cycle. Here Theorem 1 does not apply as row 3 is not the highest row being considered in this iteration. The candidate in this case is (3,1) with value 10. Now

$$22 + 7 + 10 < 40$$

and the partial solution does not produce a cycle hence ψ is augmented to

$$\psi = \{(1,2), (-2,-3), (2,4), (3,1)\},$$

iz = 32, and min = 0.

The candidate from row four is (4,3) with value 12. Now

$$32 + 0 + 12 > 40;$$

hence backtracking begins.

Row one is underlined and the entries to the right in ψ are dropped giving

$$\psi = \{(-1,-2)\},$$

iz = 0, and min = 17. The new candidate from row one is (1,4) with distance 8. Now

$$0 + 17 + 8 < 40;$$

hence ψ is augmented to

$$\psi = \{(-1,-2), (1,4)\},$$

iz = 8, and min = 12.

The candidate from row two is (2,3) with distance 5. Now

$$8 + 12 + 5 < 40$$

and no cycle is produced; hence ψ is augmented to

$$\psi = \{(-1,-2), (1,4), (2,3)\},$$

iz = 13, and min = 7.

The candidate from row three is (3,2) with value 5. Now

$$13 + 7 + 5 < 40$$

but (3,2) produces a cycle. Hence ψ is augmented to

$$\psi = \{(-1,-2), (1,4), (2,3), (-3,-2)\}$$

and the search continues. The new candidate is (3,1) with value 10. Now

$$13 + 7 + 10 < 40$$

and no cycle is produced hence ψ is augmented to

$$\psi = \{(-1,-2), (1,4), (2,3), (-3,-2), (3,1)\},$$

iz = 23, and min = 0.

The candidate from row four is (4,2) with value 7. Now

$$23 + 0 + 7 < 40$$

and ψ is a new "current best solution." This solution is stored in ω as

$$\omega = \{(1,4), (2,3), (3,1), (4,2)\}$$

and 30 becomes the new bound. The new working vector ψ is

$$\psi = \{(-1,-2), (1,4), (2,3), (-3,-2), (3,1), (4,2)\}.$$

Each row, if it is possible to have one, has one underlined element so the first iteration is complete. The new working vector becomes ω, or

$$\psi = \{(1,4), (2,3), (3,1), (4,2)\}$$

and the second iteration begins.

Backtracking continues by requiring that two elements in each row be underlined. As before, the last two row assignments are set free and the assignment for row two is underlined giving

$$\psi = \{(1,4), (-2,-3)\},$$

iz = 8, and min = 12.

The candidate from row two is (2,1) with value 15. Now

$$8 + 12 + 15 > 30$$

is the new "current best solution." Thus backtracking continues to the first row giving

$$\psi = \{(-1,-4)\},$$

iz = 0, and min = 17.

The candidate from row one is (1,3) with value 10. Now

$$0 + 17 + 10 < 30$$

and ψ is augmented to

$$\psi = \{(-1,-4),\ (1,3)\},$$

iz = 10, and min = 12.

The candidate from row two is (2,4) with distance 7. Now

$$10 + 7 + 12 < 30$$

and there is no cycle; hence ψ is augmented to

$$\psi = \{(-1,-4),\ (1,3),\ (2,4)\},$$

iz = 17, and min = 7.

The candidate from row three is (3,2) with value 5. Now

$$17 + 7 + 5 < 30$$

and no cycle is produced; hence ψ is augmented to

$$\psi = \{(-1,-4),\ (1,3),\ (2,4),\ (3,2)\},$$

iz = 22, and min = 0.

The entry from row four is (4,1) with value 8. Now

$$22 + 0 + 8 \geq 30,$$

so backtracking begins in row three and

$$\psi = \{(-1,-4),\ (1,3),\ (2,4),\ (-3,-2)\},$$

iz = 17, and min = 7.

The new candidate from row three is (3,1) with value 10.

Now

$$17 + 7 + 10 > 30$$

hence backtracking continues to row two giving

$$\psi = \{(-1,-4),\ (1,3),\ (-2,-4)\},$$

iz = 10, and min = 12.

The candidate is (2,1) with value 15. Again

$$10 + 12 + 15 > 30$$

so backtracking continues giving

$$\psi = \{(-1,-4),\ (-1,-3)\},$$

iz = 0, and min = 17.

The candidate is (1,2) with value 15. Again

$$0 + 17 + 15 > 30$$

and iteration 2 is complete. As n-2 iterations have been made all solutions have been enumerated, (see Theorem 2). The minimum solution is therefore

$$\omega = \{(1,4),\ (2,3),\ (3,1),\ (4,2)\}$$

with value 30.

C. Shortest Route

To use the algorithm for a shortest route problem

1. Read the first city to be visited as (x_1,y_1) and the final city to be visited as (x_n,y_n).
2. After the matrix D has been generated set $d_{1,n} = d_{n,1} = 0$.

This can be implemented through a code on an input card.

D. Comments

It is noted that only row k needs to be in memory at any time; therefore matrix D can be generated as needed or stored in auxiliary memory.

The dimension of ψ has an upper bound of $2(n-1)^2$ where n is the number of cities. However it was found for the problems considered that substantially less memory was required. This is due to the bounding technique of the algorithm. A compromise between the number of rows of D in memory and the dimension of ψ is dictated by the size of the problem and the memory size of the computer.

It appears that Step 2a and 2c of the algorithm are the same. However, for purposes of coding it is necessary to distinguish between the two branches. In each case the number of elements set free in ψ is computed from different formulas.

The search for the minimum free element differs with the values of α. If α_k is 0 or 1 the search is carried to the right of the diagonal in row k. In order to make this true when increasing the number of underlines required a new D matrix (Step 6) is generated when this increment is made. Note that this change was not made in the sample problem.

Another property of the algorithm is that a "current best solution," along with the number of underlines required, can be punched out and read in at a later time. The algorithm will continue to iterate at that point. This locates the starting point better than Balas' algorithm which would use only the "current best solution."

As the example illustrates all solutions could be generated by a small alteration of the algorithm. Also it appears that the algorithm can be used for the assignment problem by eliminating the steps where cycling is checked.

There are several ways of terminating the computation. For example:

1. After all solutions have been enumerated.
2. For a fixed value of ns.
3. When there is no change in z from one iteration to the next.
4. When the relative change in z between iterations is "small."
5. When the difference between the "current best solution" and the lower bound is "small." (See Chapter VI).

Finally, note that the algorithm is not a combination or revision of any known algorithm. However the algorithm tends to incorporate some of the ideas of other algorithms but in an entirely different manner.

V. COMPUTATIONAL EXPERIENCE

A new algorithm for the traveling salesman problem has been presented with the characteristics described in the introduction. Computational results and comparisons with Lin's 3-opt algorithm are contained in this chapter. The algorithm was implemented on an IBM-1130 with 8k words of memory. All computation work referred to in this chapter was performed on this machine.

A. Test Problems

The algorithm was tested extensively on the following five problems. These problems were designed so that solution improvement could be followed and for which final solutions were readily available. In addition comparisons were made with Lin's 3-opt algorithm for these problems. Both algorithms were given the same initial routes with initial distances as given in column 6 of Table I. The test problems follow:

1. 6-city with (x,y) coordinates (0,0), (5,5), (8,2), (10,0), (8,8), (10,10), (0,10);
2. 10-city with (x,y) coordinates (0,0), (5,5), (8,2), (10,0), (8,8), (10,10), (0,10), (5,8), (2,4);
3. 10-city with (x,y) coordinates (0,0), (10,2), (2,10), (0,8), (1,0), (10,6), (7,8), (0,5), (6,0), (10,8);
4. 15-city with (x,y) coordinates (0,0), (5,5), (8,2), (10,0), (8,8), (10,10), (0,10), (5,8), (2,8), (2,5), (0,6), (5,3),

(9,5), (4,1), (0,2);

5. 16-city with (x,y) coordinates (0,0), (20,1), (7,0), (20,0), (16,25), (0,23), (3,0), (17,10), (9,24), (1,19), (5,1), (17,15), (6,24), (1,12), (12,0), (17,19).

A summary of the results is contained in Table I.

TABLE I

COMPARISON WITH LIN'S ALGORITHM FOR TEST PROBLEMS

Problem Numbers	Algorithm Time	Algorithm Distance	3-Opt Time	3-Opt Distance	Actual Minimums	Initial Distance
(1)	.002	46	.002	44	44	54
(2)	.003	58	.007	58	54	64
(3)	.014	54	.005	78	40	112
(4)	.050	66	.024	76	60	90
(5)	.066	108	.030	280	100	348

All times were taken from the console clock of the IBM-1130 with the aid of pause statements. Time is measured in hours on the 1130.

Results for additional iterations of the algorithm are contained in Table II.

The above test problem indicates that the algorithm is competitive with the 3-opt algorithm. Only for problem 1 was the 3-opt results better than that of the algorithm, and then but slightly (2 units). For problem 5 the algorithm gave a significant improvement over the 3-opt (172 units). Also Table II indicates that the algorithm tends to converge on the second or third iteration. Because these problems were chosen for easy testing it was decided to generate ten random problems and make the comparisons again.

TABLE II

RESULTS OF FURTHER ITERATIONS FOR TEST PROBLEMS

Problem	NS = 2		NS = 3		NS = 4	
Number	Time	Distance	Time	Distance	Time	Distance
(1)	.003	44	.002	44	.002	44
(2)	.012	54	.018	54	.022	54
(3)	.006	46	.006	40	.002	40
(4)	.098	60	.078	60	.088	60
(5)	.200	100	.148	100	.211	100

B. Random 10-City Problems

Ten 10-city problems were randomly generated.* These were also compared with Lin's 3-opt as well as used to determine other statistical data. In Tables III and IV comparisons are given which parallel those given in Tables I and II. Table III is given in the form of a distance table and Table IV is given as a time table.

Again the algorithm is competitive with the 3-opt algorithm for the first iteration. Only for two of the ten problems does

* See Appendix I for a list of the problems.

3-opt give a better solution, (problems R10-10, and R10-4), and this improvement is relatively small. Note however, that all solutions given by the second iteration are better than those given by 3-opt. The actual minimums were obtained by using all iterations. However the algorithm's solution is significantly better for several problems; see, for example, problems R10-2, R10-5, R10-8, and R10-9. The time saving achieved by using the 3-opt instead of the algorithm for the first iteration is insignificant.

TABLE III

DISTANCE COMPARISONS

Problem	Initial	3-Opt	Algorithm Distance by Iteration				Actual
Number	Distance	Distance	1	2	3	4	Minimum
(R10-1)	512	384	308	286	286	286	256
(R10-2)	294	204	134	130	130	130	130
(R10-3)	290	196	188	184	184	184	184
(R10-4)	334	274	280	258	236	236	236
(R10-5)	500	352	262	258	258	258	224
(R10-6)	432	316	282	246	246	246	246
(R10-7)	496	382	332	306	306	306	290
(R10-8)	510	350	258	258	258	258	254
(R10-9)	598	446	250	224	224	224	224
(R10-10)	384	290	328	264	240	240	236

An advantage of the algorithm is that the algorithm can continue in an iterative process. Tables II and III give the results for continuation for the fifteen problems. Of the fifteen problems eight converged to the exact answer on the second iteration and two converged to the exact answer on the third iteration.

This leads one to conjecture that if one wanted the exact solution in the shortest time ns could be set to ns = n-1 after the first iteration. That is, the iterations where 2, 3, ..., n-2 underlines are required could be eliminated. This conjecture was tested on problems R10-1 and R10-2 and resulted in a decrease in time of 11%.

TABLE IV

TIME COMPARISONS

Problem	3-Opt	Algorithm Time by Iteration			
Number	Time	1	2	3	4
(R10-1)	.008	.010	.018	.026	.048
(R10-2)	.006	.006	.010	.022	.042
(R10-3)	.010	.010	.032	.076	.168
(R10-4)	.006	.008	.048	.084	.122
(R10-5)	.008	.008	.018	.040	.088
(R10-6)	.008	.012	.040	.084	.172
(R10-7)	.006	.006	.034	.118	.172
(R10-8)	.008	.006	.012	.024	.048
(R10-9)	.006	.006	.018	.040	.090
(R10-10)	.008	.008	.052	.078	.144

C. Bound Tests

Using the above problems the bound technique was tested. That is, the number of the (n-1)!/2 solutions that were explicitly enumerated were counted. The results follow in Table V. Comparison with the 3-opt is meaningless except to note that the 3-opt seems to explicitly enumerate more solutions than the first iteration of the algorithm. Table V indicates that only a fraction of the 181,440 solutions are explicitly enumerated.

TABLE V

NUMBER OF SOLUTIONS EXPLICITLY ENUMERATED

Problem	The Algorithm by Iteration			Lin's
Number	1	2	3	3-Opt
(R10-1)	9	1	0	11
(R10-2)	9	1	0	12
(R10-3)	6	1	0	14
(R10-4)	4	5	3	9
(R10-5)	7	1	0	9
(R10-6)	9	3	0	9
(R10-7)	4	4	0	12
(R10-8)	9	0	0	7
(R10-9)	6	5	0	8
(R10-10)	4	5	1	8

D. Convergence Time Tests

Three five-city and three 15-city problems were randomly generated. These, in addition to the ten-city problems and the problems of section E were used to test computer time with respect to the number of cities. Tables VI and VIII gives the results of these tests. The number following the R is the number of cities. A formula for computer time as a function of the number of cities is given in section F of this chapter.

TABLE VI

TIME VS. THE NUMBER OF CITIES

	Problem Number	Iteration 1		Iteration 2	
		Time	Distance	Time	Distance
Five Cities	(R5-1)	.002	254	.004	254
	(R5-2)	.002	220	.004	220
	(R5-3)	.002	224	.003	224
Ten Cities	(R10-1)	.010	308	.018	286
	(R10-2)	.006	134	.010	130
	(R10-3)	.022	184	.066	184
15 Cities	(R15-1)	.096	280	.190	274
	(R15-2)	.060	352	.680	302
	(R15-3)	.126	284	1.054	260

To test the practicality of generating the distance matrix as needed we used the same problems as those referred to in Table VI. Here the row distances were computed, as needed, by the formula given in section B of Chapter III.

TABLE VII

D MATRIX IN STORAGE VS. DISTANCE GENERATED

	Problem Number	Iteration 1 D Generated	 D In	Iteration 2 D Generated	 D In
Five Cities	(R5-1)	.002	.002	.003	.002
	(R5-2)	.002	.002	.004	.002
	(R5-3)	.003	.003	.005	.001
Ten Cities	(R10-1)	.014	.010	.020	.008
	(R10-2)	.006	.006	.036	.004
	(R10-3)	.010	.022	.018	.044
15 Cities	(R15-1)	.108	.096	.212	.096
	(R15-2)	.066	.066	.750	.620
	(R15-3)	.136	.126	1.026	.928

Thus it appears that for problems under size 15 the first version is slightly superior to the second. The 15-city problem indicates that time is going to increase significantly when generating the D matrix. The erratic behavior of the timings is explained by the fact that the number of times the distances are generated is dependent on the number of solutions explicitly enumerated. With the first version we are able to make three iterations for a 40-city problem on an IBM-1130 with 8k memory; that is, without using auxiliary memory. By generating the distances up to three iterations can be made for a 200-city problem on this machine.

E. Circuit Board Problems

Three live problems were taken from the printed circuit industry. In this industry a route is generated by a person using a magnifying glass and drawing a route on the photograph of the circuit board. From this sketch a paper tape which controls the drilling machines is prepared by a well trained technician. It was from these tapes that our data was prepared. In Table VIII the results for three problems are given. Table VIII shows that the technician makes both "good" and "poor" first approximations. The actual minimums are not known for these problems. The (x,y) coordinates for these problems are located in the appendix.

TABLE VIII

PRINTED CIRCUIT PROBLEMS

n	Visual Distance	3-Opt Time	3-Opt Distance	Algorithm Time	Algorithm Distance
20	79	.10	77	.32	77
28	3728	.36	3028	1.24	2164
40	152	1.57	144	2.14	144

For these examples the savings over the visual distance range from less than 3% to almost 50%. In the next section a decision function is presented which incorporates the computational experience given above and which will aid in determining when to use the algorithm.

F. Economic Considerations

The above sample problems indicate that it may not be economically feasible to use the algorithm. To help in this determination a decision function was developed. First it is noted that all circuit boards are different. Some boards contain extremely complex and dense circuitry with randomly placed holes while others are quite uniform in nature. A board may have several different hole sizes and each size must be treated as a separate problem. Circuit boards range in size from 12" by 18" to less than 1" by 1". All circuits are shown on the photograph of the board from which the tape programmer prepares a tape for the tape drill machine. All of these factors effect the tape programmer's choice of routes and helps account for the unreliable estimates.

If the tape programmer is eliminated and random tours are used as initial estimates, Tables III and IV show that the algorithm still performs quite satisfactorily.

The decision function will only be an aid to the engineer as many variables which the engineer must consider are not contained in the function. For example, the experience of the programmer and the availability of computer and/or tape drill time are not considered by the function.

Note that there are several types of tape drills and consider the following definitions:

1. N as the number of boards to be produced.
2. n the number of holes per board.
3. d the distance obtained from the tape prepared by the tape programmer.
4. sh the stack height, that is, the number of boards that are drilled by one drill bit. The maximum is five.
5. nh the number of heads on the drill, all of which are controlled simultaneously by the tape. The maximum is six.
6. ms drill movement speed.
7. cp the computer cost per hour.
8. dc the tape drill cost per hour.
9. ad approximate distance reduction.

The algorithm would be recommended for use if the following function is positive,

$$F(N,n) = dc \cdot g(n) - cp \cdot h(n)$$

where

$$g(N) = ad(d/ms)\ (N/sh \cdot nh)$$

and

$$h(n) = .0654n - .6497$$

The coefficients for h were found by fitting the data of Tables VI and VIII to the curve using the least squares method. Of a cubic, quadratic and linear approximation, the linear gave the best fit. Note that the formula is for the first iteration of the algorithm only. For computational purposes we use cp = 2dc for a six-headed drill where cp is the cost related to the IBM-1130.

Computations with conservative figures show that the number of boards one must produce in order to break even are approximately 12,000, 2,500, and 60,000 for the 20, 28, and 40 hole board respectively. Orders of around 20,000 boards are not uncommon and these orders are sometimes repeated. For such an order the savings made possible by the algorithm for the 28 hole board would be in three figures.

G. Shortest Route Results

The algorithm was not tested extensively for this problem. However, this formulation was checked for problems 1-5 with satisfactory results. Also the 20-city problem listed above was solved as a shortest route problem. The times for problems 1-5 were slightly less than those reported in Tables I and II.

VI. CONCLUSION

A. Summary

The first known coordinate oriented algorithm was developed and presented in Chapter III. The algorithm was developed for a particular problem in the printed circuit industry. However, it can be utilized, with only minor modifications, for all traveling salesman problems for which coordinates are available.

Chapter II presents a review of the literature with emphasis placed on those algorithms applicable to the problem as given in Chapter I. In the discussion of Chapter IV the similarities and differences of these algorithms and the one presented in Chapter III are pointed out. These comparisons serve to emphasize that the algorithm presented is indeed new.

In Section E of Chapter V the algorithm is applied to three problems from the printed circuit industry. The results vary widely with the problems. These variations are explained in section F of Chapter V where an economic decision function is presented.

The theorems presented in Chapter III and the computation results given in Chapter IV indicate the findings given in the next section are valid.

B. Results

1. The performance of the algorithm for the first iteration is superior to that of the best known algorithm (3-opt) for a locally optimal solution. Of the 24 problems solved, the 3-opt algorithm gave slightly better results at the end of one iteration for only 3 of the problems.

2. The algorithm converges to the exact solution, (see Theorem 2 of Chapter III).

3. The algorithm can be stopped at any time giving a "current best solution." Also this solution can be used later as an initial solution for the algorithm, (see Chapter III, Section C).

4. The algorithm is easily adaptable to a shortest route problem. Section C of Chapter IV contains the formulation and Section G of Chapter V gives the computational experience.

5. The algorithm can be effectively implemented on a small computer. All work was done on an IBM-1130 with 8k words of memory.

6. The distance matrix can be generated as needed, thus increasing the size of the problem which can be solved.

This ability was demonstrated in Section D of Chapter V.

7. The bound criterion is successful as only a few of the solutions are enumerated explicitly. Experimental results for ten problems is given in Section C of Chapter V.

8. The algorithm generates all constraints from the coordinates and the solution is given in coordinates, (see Chapter III).

C. Suggestions for Further Research

The codes for the algorithms were programmed in FORTRAN with all variables integers. The programs consist almost entirely of additions, subtractions, and comparisons. As the programs for the algorithm have undergone constant revision the codes used for timings were undoubtedly less than optimal.

There are two other areas where the speed of convergence of the algorithm might be improved.

1. The cycle test (II,4) of the 0-1 integer programming formulation given in the introduction was utilized in the codes. The cycle constraints (II,7) of Tucker should also be tested.

2. The bounding technique might be improved, at least for large problems, by adding to the distance of the partial

solution the sum of the minimums for the "free" rows where the row minimum is the minimum of the "free" columns.

Both computation and search time might be reduced for symmetric problems by restructuring the algorithm for this problem.

A combination of the λ-opt and the algorithm might be used in an iterative process. That is, from an arbitrary solution, find a λ-opt solution and use this for the initial solution for the algorithm. The solution given by the algorithm would then be used as input for the λ-opt algorithm. This process could continue as long as improvement was made.

Little's algorithm might be used in conjunction with the algorithm presented here to help isolate the actual minimum. This could be done by using the algorithm to find a decreasing sequence of upper bounds on the solution and Little's algorithm to find an increasing sequence of lower bounds. The procedure could be terminated when these bounds were relatively close; i.e., when the difference between the upper bound and lower bound divided by the upper bound is small.

The algorithm could also be modified and tested for solutions to the assignment problem.

Finally, the author feels that the algorithm can be adapted to a partitioning technique for "large" problems, and it is in this area that his future efforts will be directed.

APPENDIX A

COORDINATES FOR SAMPLE PROBLEMS

Problem (R5-1)

(10,3), (21,15), (0,55), (20,80), (34,96).

Problem (R5-2)

(22,7), (28,81), (19,31), (37,96), (16,80).

Problem (R5-3)

(25,0), (43,73), (10,48), (15,48), (27,79).

Problem (R10-1)

(24,22), (6,94), (18,75), (10,17), (24,53), (22,74), (30,0), (25,10), (12,33), (30,75).

Problem (R10-2)

(27,61), (19,61), (26,39), (13,49), (29,59), (24,40), (30,78), (17,36), (32,75), (25,79).

Problem (R10-3)

(13,44), (12,46), (8,89), (11,54), (30,31), (5,34), (22,30), (23,55), (14,61), (30,34).

Problem (R10-4)

(25,49), (29,54), (14,42), (40,39), (21,27), (11,2), (29,38), (29,75), (13,86), (28,34).

Problem (R10-5)

(33,46), (22,43), (27,5), (27,60), (20,86), (8,2), (41,65), (22,91), (14,28), (20,51).

Problem (R10-6)

(1,30), (25,56), (22,73), (7,57), (41,99), (17,67), (16,18), (22,56), (20,85), (17,30).

Problem (R10-7)

(0,95), (38,60), (22,99), (14,1), (10,48), (11,55), (30,96), (17,61), (9,47), (17,78).

Problem (R10-8)

(4,73), (9,65), (40,10), (22,25), (27,97), (29,19), (32,79), (19,68), (31,47), (22,97).

Problem (R10-9)

(35,46), (19,90), (25,42), (20,30), (2,88), (25,36), (21,12), (26,81), (17,12), (14,85).

Problem (R10-10)

(11,47), (38,81), (26,61), (11,82), (12,50), (14,21), (20,14), (25,49), (41,94), (24,55).

Problem (R15-1)

(5,15), (36,27), (21,68), (29.70), (10,3), (15,79), (19,41), (33,37), (25,14), (32,43), (30,47), (23,19), (23,83), (23,33), (27,7).

Problem (R15-2)

(26,63), (21,17), (32,80), (21,44), (24,52), (30,6), (36,70), (23,94), (22,74), (24,3), (12,65), (13,32), (17,92), (25,10), (3,46).

Problem (R15-3)

(37,22), (17,32), (22,37), (32,90), (9,35), (18,67), (17,98), (27,52), (7,85), (25,44), (26,52), (8,83), (26,36), (28,66), (23,98).

Problem (20-City)

(47,7), (49,11), (46,11), (46,13), (47,12), (43,12), (45,14), (50,14), (56,17), (58,16), (60,16), (49,18), (47,18), (43,18), (46,19), (45,20), (48,20), (47,21), (43,23), (42,25).

Problem (28-City)

(17,825), (42,825), (202,825), (227,825), (387,825), (412,825), (572,825), (597,825), (758,825), (783,825), (923,825), (737,825), (552,825), (367,825), (17,877), (42,877), (202,877), (227,877), (387,877), (412,877), (572,877), (597,877), (758,877), (783,877), (923,877), (737,877), (552,877), (367,877).

Problem (40-City)

(16,11), (19,12), (20,8), (21,7), (19,6), (16,5), (15,7), (12,6), (10,6), (10,9), (12,12), (10,12), (10,15), (12,18), (10,18), (10,20), (10,21), (4,23), (9,24), (11,25), (10,27), (12,28), (11,30), (10,30), (10,33), (10,36), (10,39), (10,42), (2,42), (2,39), (2,36), (2,33), (2,30), (2,27), (2,21), (2.18), (2,15), (2,12), (2,9), (2,6).

BIBLIOGRAPHY

1. Dantzig, G. B., Fulkerson, D. R. and Johnson, S. M., "On a Linear Programming, Combinatorial Approach to the Traveling Salesman Problem," *Operations Research*, VII, 1959, pp. 58-66.

2. Flood, M. M., "The Traveling Salesman Problem," *Operations Research*, IV, 1956, pp. 61-75.

3. Ore, Oystein, "Graphs and Their Uses," New York, *Random House* and *The L. W. Singer Company*, 1963, pp. 28-31.

4. Bellmore, M., and Nemhauser, G. L., "The Traveling Salesman Problem: A Survey," *The John Hopkins University*, Baltimore, Maryland, 1966.

5. Little, J. D. C., Murty, K. G., Sweeney, D. W. and Karel, C., "An Algorithm for the Traveling Salesman Problem," *Operations Research*, XI, 1963, pp. 972-989.

6. Miller, C. E., Tucker, A. W. and Zemlin, R. A., "Integer Programming Formulations and Traveling Salesman Problems," *Journal of the Association for Computing Machinery*, VII, 1960, pp. 326-329.

7. Gomory, R. E., "Outline of an Algorithm for Integer Solutions to Linear Programs," *Bulletin of the American Mathematical Society*, LXIV, 1958, pp. 275-278.

8. Kunzi, Hans P. and Tzschach, H. G., and Zehnder, C. A., "Numerical Methods of Mathematical Optimization," *New York Academic Press*, 1968.

9. Land, A. H. and Doig, A. G., "An Automatic Method of Solving Discrete Linear Programming Problems," *Econometrica*, XXVIII, 1960, pp. 497-520.

10. McMillan, Claude, Jr., "Mathematical Programming," New York, *John Wiley & Sons, Inc.*, 1970.

11. Balas, E., "An Additive Algorithm for Solving Linear Programs with Zero-One Variables," *Operations Research,* XIII, 1965, pp. 517-546.

12. Geoffrion, A., "Integer Programming by Implicit Enumeration and Balas' Method," *SIAM Review*, IX, 1967, pp. 178-190.

13. Kruskal, J. B., "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem," Proceedings American Mathematical Society, II, 1956, pp. 48-50.

14. Shapiro, Donald M., "Algorithms for the Solution of the Optimal Cost and Bottleneck Traveling Salesman Problems," Ph.D. Thesis, 1966, Washington University.

15. Sweeney, D. N., "Exploration of a New Algorithm for Solving the Traveling Salesman Problem," M. S. Thesis, 1963, M. I. T.

16. Lin, S., "Computer Solution of the Traveling Salesman Problem," Bell System Technical Journal, XLIV, 1965, pp. 2245-2269.

17. Croes, G. A., "A Method for Solving Traveling Salesman Problems," Operations Research, VI, 1958, pp. 791-812.

18. Miller, L. R., "Heuristic Algorithms for the Generalized Vehicle Dispatch Problem," Ph.D. Thesis, 1970 University of Missouri-Rolla.

VITA

Joseph Sidney Greene was born on February 29, 1932, at Morganton, North Carolina. He graduated from The High School Division of Warren Wilson College in June 1950. In June 1950 he entered the U. S. Navy and served for four years in the Pacific. In May 1954 he entered Southwest Missouri State earning the Bachelor of Science degree with majors in Industrial Education and Mathematics in January 1957. From January 1957 to June 1961 he was employed at The School of the Ozarks as a teacher-administrator. During this period he attended summer school at the University of Missouri-Columbia studying Administration and Mathematics. In June 1960 he received the Master of Education degree. In the summer of 1961 and academic year of 1961-1962 he received N.S.F. fellowships in Mathematics at The University of Missouri-Columbia and Oklahoma State University respectively. In August 1963 he received the Master of Science Degree in Mathematics at Oklahoma State University. In the summer of 1965 and 1968 he received N.S.F. fellowships in Computer Science at the University of Missouri-Rolla. From 1963 to the present he has been employed at Drury College as an Assistant Professor of Mathematics. Drury granted him a sabbatical for further study in 1968-1969.

In May 1968, he formed Springfield Computer Consultants, Inc., which currently employs four people.