WORKLOAD BALANCING IN VEHICLE ROUTING PROBLEMS

By

JERRY DENVER ALLISON

Bachelor of Science
University of Texas at Arlington
Arlington, Texas
1968

Master of Engineering
Texas A and M University
College Station, Texas
1970

Submitted to the Faculty of the Graduate College
of the Oklahoma State University
in partial fulfillment of the requirements
for the Degree of
DOCTOR OF PHILOSOPHY
December, 1986

WORKLOAD BALANCING IN VEHICLE

ROUTING PROBLEMS

Thesis Approved:

Marvin Palmer Terrell
Thesis Adviser

Joe H. Mize

Allen C. Schuermann

Michael H. Branson

Donald W. Grace

Norman N. Durham
Dean of the Graduate College

PREFACE

This study is concerned with the vehicle routing problem (VRP) in which it is desired to minimize, in addition to the total distance over all routes, the deviation in workload among the routes. Two workload elements are considered: (1) the total distance or time spent driving, and (2) the total weight or amount of goods delivered. This problem is termed the workload balanced vehicle routing problem (WBVRP). The purpose of the study is to develop an interactive model to solve the WBVRP using multiple criteria analysis.

I wish to express my gratitude to my major adviser and chairman of my Ph.D. committee, Dr. M. Palmer Terrell, for his encouragement and guidance during this study and throughout my doctoral program. I wish also to thank the members of my committee, Dr. Michael H. Branson, Dr. Joe H. Mize, Dr. Allen C. Schuermann, and Dr. Donald W. Grace, for their interest and assistance.

I wish also to thank the staff of the University Computer Center for their help and cooperation in developing the graphics capability for the interactive program.

The love, understanding, and patience of my wife, Susan, have enabled me to complete this dissertation. Her faith and trust, and that of my two daughters, Ginger and Leslie, have helped me endure the many long and difficult hours required for this research.

TABLE OF CONTENTS

Chapter Page

Chapter Page

LIST OF TABLES

Table | Page

LIST OF FIGURES

Figure | Page

CHAPTER I

INTRODUCTION

Problem Definition

Over the past twenty-five years, the vehicle routing problem (VRP) has received the attention of many researchers. The problem has typically been solved using single-objective optimization or heuristic methods, the objective usually being the minimization of total route time or distance traveled. Problem constraints might vary, but usually include maximum route time or distance constraints, vehicle capacity constraints, and might include limits on the number of available vehicles.

One of the objectives which a distribution manager must usually consider is the equitable distribution of workload among the workers in the system. For purposes of this research, the following elements of workload are considered:

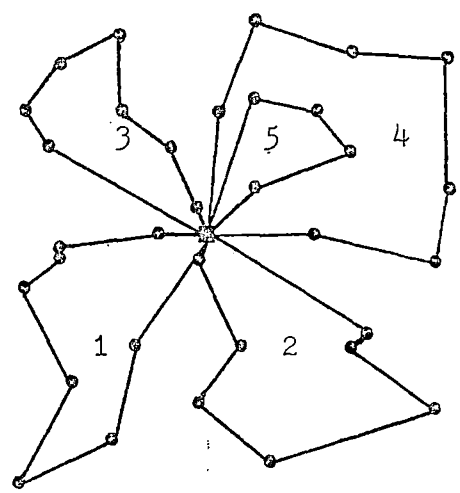
1. Total driving workload, expressed as the total distance driven or total time spent driving by a given crew.
2. Total handling workload, expressed as a total weight of goods picked up or delivered, or total time spent in loading/unloading goods picked up or delivered, or total stops made by a given crew.

The particular units of measure for these workload elements depend upon the specific problem being solved.

The workload-balanced vehicle routing problem (WBVRP), then, is the VRP which has as its objectives, in addition to the minimization of total time or distance traveled, the equitable distribution among the workforce of one or more of the elements above. By this definition, the WBVRP is seen to be a multiobjective problem.

An 'ideal' solution to the WBVRP would achieve a minimum total time or distance traveled, and would have both workload elements distributed equally among the workforce. However, because of the combinatorial nature of the problem and the tradeoffs necessary in achieving a perfectly balanced solution, this 'ideal' solution is usually not possible. Instead, the preference structure of the route planner dictates the tradeoffs among the objectives, and a compromise solution is accepted.

As an example, consider the 33-city VRP represented in Figure 1.1. The five routes shown are a minimum-distance set, the total distance over all routes being 174 miles. Note, however, that the lengths of individual routes vary from 21 miles for the shortest route (route number five) to 43 miles for the longest route (route number four). Also, note that the load carried on the lightest route (route number three) is 1190 pounds, while the load carried on the heaviest route (route number four) is 1460 pounds. A more balanced set of routes for the same VRP is shown in Figure 1.2. In this route set, the difference between the shortest and longest routes (routes three and two, respectively) is only seven miles, and the difference between the lightest and heaviest routes (routes one and four, respectively) is only 90 pounds. Note, however, that the total distance traveled over the routes in Figure 1.2 is 186 miles. The difference of 12 miles in total distance



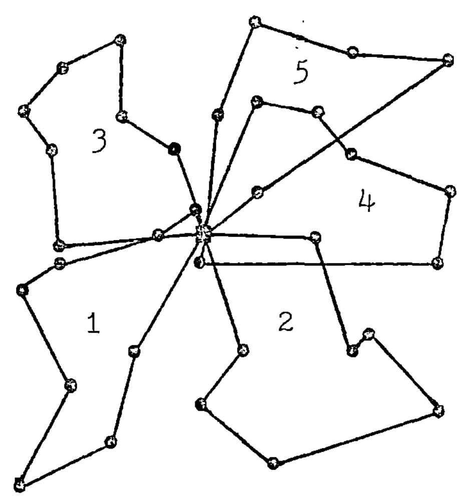
Route	Load	Length
1	1410	40
2	1300	39
3	1190	31
4	1460	43
5	1220	21

Total Distance = 174

Route Load Deviation = 270

Route Length Deviation = 22

Figure 1.1. Minimum Distance Solution to 33-City Vehicle Routing Problem



Route	Load	Length
1	1260	39
2	1320	41
3	1340	34
4	1350	36
5	1310	36

Total Distance = 186

Route Load Deviation = 90

Route Length Deviation = 7

Figure 1.2. Workload Balanced Solution to 33-City Vehicle Routing Problem

between Figures 1.1 and 1.2 represents the penalty which must be paid to achieve the level of balance shown.

The WBVRP can manifest itself in several different ways. Some delivery crews consist of a driver and a helper, the helper's duty being primarily that of loading or unloading the goods at each stop. A set of routes which exhibits a wide variance in the helper's workload, even if the drivers all have approximately the same amount of driving workload, is not an acceptable set, at least to the helpers who must do the greater share of the handling workload. On the other hand, a set of routes which balance the helpers' workload while exhibiting a wide variance in the drivers' workload is not acceptable to some of the drivers. In trying to achieve an acceptable solution, the route planner must weigh the importance of management's objective, usually the minimization of cost, against the competing claims of different sectors of the delivery workforce, and make appropriate tradeoffs.

Even with crews which consist of only a driver, workload balancing is an important issue. Here the driver does all the work, so the route planner does not have different workforce skills to consider. He does, however, have the problem of trading off different levels of the workload elements against each other and against the total distance or time objective. The problem exists regardless of the perceived relative desirability of the different workload elements to the workforce. In one situation, the driving element might be preferred over the handling element, especially if the handling element requires a large amount of heavy lifting and/or carrying effort. In another situation, the handling element might be preferred over the driving element. An example of this would be the case in which adverse driving conditions exist. The important thing is not the relative desirability of one workload element versus the other, but the fact that preferences do exist, necessitating that tradeoffs be made among them.

Routing problems closely related to the WBVRP are those in which the loads carried by the vehicles do not necessarily represent one of the workload elements above, *per se*, but in which it is nonetheless important to balance the vehicle loads. An example of this is the VRP in a driver-sell environment, in which at least part of the driver's earnings are dependent upon the quantity of goods delivered. Another example is the school-bus routing problem in which it is desired to equalize to some extent the number of children carried in the different buses, thereby distributing the responsibility for the children's safety among the bus drivers. Because such problems can be formulated in the same manner as the WBVRP, they can be considered to be included in the overall scope of this research.

Advantages of Workload Balancing

A set of workload-balanced routes has several potential advantages over a minimum-distance or minimum-time set of routes. These advantages fall within four areas.

Employee Relations

For most companies, the primary reason for using workload-balanced routes is that the perception of workload equity by the workforce should enhance employee morale. This is turn can lead to fewer worker complaints, lower levels of absenteeism, greater levels of cooperation, and a general trend toward higher labor productivity.

Crew Scheduling

Company and/or union goals sometimes specify certain levels of workload equity over a period of time. If the crews are always assigned to workload-balanced routes, there is no need to resort to sophisticated crew scheduling systems in order to attain these goals.

Route Stability

Fixed routes are sometimes desired. However, if customer demand changes over time, these routes will eventually have to be altered to meet vehicle capacity and route length constraints. A set of balanced routes requires fewer alterations over a long period of time than does a minimum-distance or minimum-time set of routes (Dileepan, 1984).

Fleet Management

Using workload-balanced routes, all vehicles in the fleet undergo approximately the same mileage and vehicle loads. Therefore, many fleet management decisions related to a particular vehicle will also apply to the fleet as a whole. Included are vehicle replacement intervals, preventive maintenance schedules, and intervals for tire rotations, vehicle inspection, oil changes, and so forth.

Measures of Workload Imbalance

There are several measures that could be related to the two workload elements listed on page one. The two measures which are used in this research are discussed below.

Route Length Deviation

Route length deviation is a measure of the first workload element. It is defined as the maximum route length in a route set minus the minimum route length in the set. 'Route length' can be interpreted to mean either distance traveled or time spent in traveling over a route. If applicable, the route lengths include 'drop allowances' at the various stops, measured in the same distance or time units.

Route Load Deviation

Route load deviation, a measure of the second workload element, is defined to be the maximum route load carried in a route set minus the minimum route load carried. 'Route load' is usually thought of as the weight of goods carried on a route. It may, however, be defined as the volume of product, number of items carried, number of stops on a route, or the number of customers carried (e.g., in a bus routing problem). The route loads correspond to the total demand of all stops on a route.

It could be argued that other measures of workload imbalance are appropriate. For instance, the standard deviation, average deviation, sum of squared deviations, or sum of absolute deviations could be used. Certainly, from a distribution manager's viewpoint, these measures might be adequate. Also, from a computational standpoint, the sum of squared deviation can be easily determined in 'pairwise exchange' heuristics (Dileepan, 1984). However, since employee morale is considered to be one of the primary reasons for using workload-balanced routes, and since the measures defined above should be more meaningful to members of the distribution workforce, those measures are the ones which have been adopted for this research.

Goals and Objectives of Research

Objectives

There are two primary objectives for this research. They are:

1. Examine the use of multicriteria analysis in developing vehicle routes which offer an equitable distribution of workload among the workforce.
2. Determine the implications of workload balancing to distribution management under differing conditions of customer demand and location.

Goals

To meet the objectives above, four specific goals are delineated. They are:

1. Develop a multiobjective model structure to solve the WBVRP, utilizing user interaction to make tradeoffs among the three objectives of the problem.
2. Develop and evaluate methods to minimize each of the three objective functions of the WBVRP.
3. Incorporate the multiobjective model structure into an interactive computer program, and evaluate the performance of the program in terms of efficiency (solution times) and effectiveness (solution values).
4. Solve different WBVRP problems which vary in their customer demand patterns (i.e., constant demand vs variable demand) and in the relationship between depot and customer locations. Determine the penalty, in overall time or distance units, which

distribution managements must pay in order to balance the two workload elements under these conditions.

Research Outline

The research into workload balancing in VRPs is contained in the six remaining chapters. Chapter II contains a review of the literature covering the vehicle routing problem and route balancing in VRPs. Chapter III covers the development of an interactive multiobjective model structure, based on the Method of Satisfactory Goals (Benson, 1975), which can be used to solve the WBVRP. In Chapter IV, three different single-objective algorithms, necessary to the implementation of the multiobjective model, are developed and evaluated in terms of their efficiency and effectiveness. Chapter V describes and demonstrates an interactive computer program which is used to implement the multiobjective model, and contains an evaluation of the program in terms of its ability to converge to a solution from different starting points. In Chapter VI, the solution results of several problems which vary in customer demand and location patterns are presented, and implications for distribution management are offered. Finally, Chapter VII contains conclusions from this research and offers suggestions for further research in this area.

CHAPTER II

LITERATURE REVIEW

This chapter contains a review of research in the area of vehicle routing. Because of the importance of the traveling salesman problem (TSP) in many approaches used to solve the vehicle routing problem (VRP), the chapter begins with a review of work done toward solving the TSP. Next, approaches taken to solve the VRP are reviewed. Finally, research efforts in the area of workload-balanced vehicle routes are covered.

The Traveling Salesman Problem

The TSP, generally credited to Professor Hassler Whitney of Princeton University in 1934 (Flood, 1956), can be stated as follows: A salesman wishes to leave his home and visit each of n-1 cities only once, then return. If the cost of traveling between city i and city j is c_{ij}, the salesman wishes to minimize the sum of the travel costs, Σc_{ij}. If $c_{ij}=c_{ji}$ for all i and j, the problem is said to be symmetric; otherwise, it is said to be asymmetric. The TSP is one of the most extensively researched problems in operations research. Approaches to solving it have been surveyed by Bellmore and Nemhauser (1968), Eilon et al., (1971), Christofides (1979), and Lawrence (1981).

The TSP is NP-complete. As such, there are no known algorithms which will obtain an exact solution to the problem in polynomial time. For large problems, therefore, a heuristic approach is usually taken. In the following paragraphs, both approaches are presented. Exact solution procedures are covered first, followed by heuristic procedures.

Exact Procedures

Dynamic Programming. Both Bellman (1962) and Held and Karp (1962) applied the principle of optimality to solve the TSP using dynamic programming. The major difficulty with this approach is that the storage requirements for the recursive relationships grow exponentially as the number of cities in the problem is increased. Held and Karp solved a 13-city asymmetric problem, which was the limit of their computing capacity (an IBM 7090 with 32K memory). However, Bellmore and Nemhauser (1968) demonstrated that a machine of this size could solve an 18-city problem by using auxiliary storage and a judicious selection of the values to be maintained in memory.

Integer Programming. Dantzig, Fulkerson, and Johnson (1954) solved a 42-city TSP by using a linear programming formulation. Their method avoided the large number of loop constraints (necessary to prevent subtours) by beginning with only a limited number of constraints and then adding additional ones as necessary. Fractional solutions were eliminated by applying combinatorial arguments on an ad-hoc basis. Miller, Tucker, and Zemlin (1960) used a similar approach but employed Gomory's cutting plane algorithm instead of the combinatorial arguments of Dantzig et al. Martin (1963) solved the 42-city problem using cutting planes and a different set of loop constraints which proved to be efficient in eliminating fractional solutions as well. Miliotis (1976) reported solving symmetric problems involving up to 64 cities by employing an algorithm which first achieved integrality through cutting planes, then added loop constraints as necessary.

Branch and Bound. The majority of exact solution approaches to the TSP have utilized some form of branch-and-bound procedure. There are two general approaches: (1) tour building and (2) subtour elimination. The first of these, the tour-building approach, was developed by Little et al. (1963), and used a penalty method for determining the branching process. Incidentally, their article marked the first appearance of the expression "branch and bound" in the literature.

The second approach, subtour elimination, was developed by Eastman (1958) and modified by Shapiro (1966) and Bellmore and Malone (1971). Solving an assignment problem provides an initial lower bound and, if no subtours are present, the optimal solution. If subtours are present, then changes to the cost matrix are made to prevent them from occurring in further solutions to the assignment problem.

Christofides (1972) has shown that the tightness of the lower bounds in a branch-and-bound scheme is of more importance than the branching process in determining the effectiveness of the method. In order to improve the bounds over those obtained by previous branch-and-bound procedures, he developed an algorithm which utilizes two transformations of the original cost matrix: (1) a "contraction", which is defined as the replacement of a subtour by a single node, and (2) a "compression", which is defined as the transformation of a matrix which does not satisfy the triangularity condition of metric space into one that does. The triangularity condition is met if the distance between two points is not greater than the distance between the same two points while passing through an intermediate third point. His algorithm proceeds as follows:

Step 1. Set matrix M equal to initial distance matrix. Set lower bound L equal to zero.

Step 2. If triangularity condition is met, go to Step 3. Otherwise, COMPRESS M.

Step 3. Solve assignment problem using matrix M and increase L by this amount.

Step 4. CONTRACT matrix M by replacing subtours by a single node.

Step 5. If matrix M is now 1 x 1, go to Step 6. Otherwise, go to Step 2.

Step 6. End. Lower bound = L.

Using this procedure, bounds which were on the average within 4.7 percent of optimality for symmetric TSPs and within 3.8 percent for asymmetric TSPs were obtained at an average computation premium of only 9 percent increase in time over the original assignment problem.

Balas and Christofides (1981) used the assignment-problem approach to calculating lower bounds in a subtour-elimination scheme, but their method involves the introduction of violated subtour-elimination constraints into the objective function via Lagrangean relaxation techniques. Their approach has been found to provide extremely tight bounds (within one-half percent of the TSP optimum) for asymmetric TSPs. Balas and Christofides have used this Lagrangean approach to solve asymmetric problems of up to 325 cities optimally.

Held and Karp (1970, 1971) developed a Lagrangean relaxation approach to the symmetric TSP involving bounds from minimum spanning trees (1-trees in particular). A minimum 1-tree is comprised of the minimum spanning tree through vertices 2, 3, 4, . . ., n plus the two minimum-weight arcs connecting vertex 1 to the remainder. If the minimum 1-tree is a tour, then the tour is a solution to the TSP. If the minimum 1-tree is not a tour, it does provide a lower bound on the TSP solution.

The minimum 1-tree problem can be formulated as a relaxation to the TSP. Then, by including those constraints directly in the (1-tree) objective function with their associated Lagrange multipliers, a lower bound is obtained. The greatest such bound, obtained by using the best values of the multipliers, is used in the branch-and-bound procedure. Held and Karp (1971) used an iterative subgradient optimization procedure to determine the optimal values of the multipliers, and Hansen and Krarup (1974) later presented improved methods for doing so. Using this bounding technique, symmetric problems of up to 100 cities (Christofides, 1979) have been solved.

Heuristic Procedures

Because the TSP is NP-complete, problems of large size are usually solved by a heuristic approach. These approximate methods can generally be placed into one of three different categories: (1) tour-building heuristics, (2) tour-improvement heuristics, and (3) composite heuristics. A survey of approximate algorithms for the TSP is contained in Golden et al. (1980).

Tour-building Heuristics. The most common method of building up a complete tour is through the "savings" approach. Clarke and Wright (1964) introduced this approach for the vehicle routing problem (VRP); however, if vehicles are assumed to have infinite capacity, then the resulting solution for the VRP will contain a single route (vehicle), and the solution is valid for the TSP.

The algorithm begins by linking n-1 nodes to any single arbitrary node. For convenience, call this node 1. The initial solution will therefore consist of n-1 separate subtours, each costing $c_{1i} + c_{i1}$. Now, connecting any two nodes i and j ($i,j \neq 1$) will result in the savings

$$S_{ij} = c_{1i} + c_{1j} - c_{ij} \qquad (1.1)$$

if the problem is symmetric, and

$$S_{ij} = c_{i1} + c_{1j} - c_{ij} \qquad (1.2)$$

if the problem is asymmetric. These savings are sorted in decreasing order, and subtours are formed by going down the savings list and linking the appropriate nodes i and j. Any two nodes can be linked if they are not both already in the same subtour, and if both are linked directly to node 1. Figure 2.1 illustrates the procedure. Golden et al. (1980) have shown the Clarke and Wright procedure to require on the order of $n^2 lg(n)$ computations, where lg(n) is the logarithm of n with base 2. Several modifications to the Clarke and Wright procedure have been made by others, but a discussion of these is postponed until the vehicle routing problem (VRP) is discussed. As stated previously, the Clarke and Wright algorithm was not originally developed for the TSP, but for the VRP. It is included here only because some methods, such as the tour-improvement algorithms, require an initial tour to begin with, and the savings algorithm is commonly employed for this purpose.

Another class of tour-building heuristics are the "sequential" approaches. The simplest of these is the "proximity" or "nearest neighbor" heuristic (Rosenkrantz et al., 1974). This method begins with any city in the problem, and connects it to the city nearest to it to

	1	2	3	4	5	6
1	∞	4	3	6	7	7
2	4	∞	2	5	7	8
3	3	2	∞	3	5	6
4	6	5	3	∞	2	4
5	7	7	5	2	∞	2
6	7	8	6	4	2	∞

(A) Distance Matrix

$S_{23} = 4 + 3 - 2 = 5$

$S_{24} = 4 + 6 - 5 = 5$

$S_{25} = 4 + 7 - 7 = 4$

$S_{26} = 4 + 7 - 8 = 3$

$S_{34} = 3 + 6 - 3 = 6$

$S_{35} = 3 + 7 - 5 = 5$

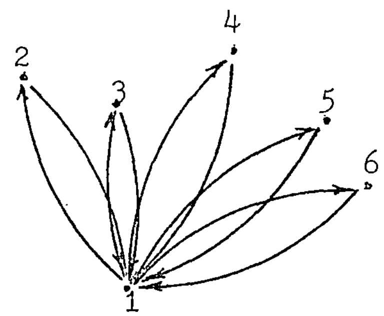
$S_{36} = 3 + 7 - 6 = 4$

$S_{45} = 6 + 7 - 2 = 11$

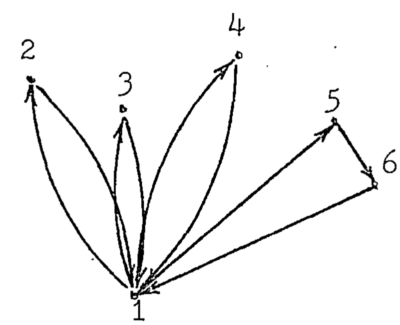
$S_{46} = 6 + 7 - 4 = 9$

$S_{56} = 7 + 7 - 2 = 12$

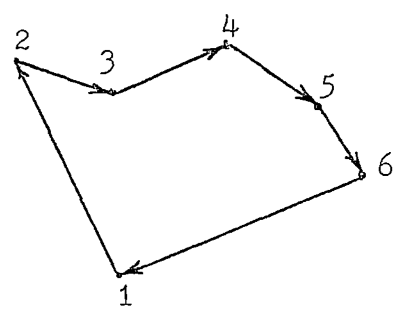
(B) Savings Calculations



(C) Initial n-1 Tours



(D) n-2 Tours After S_{56} Applied



(E) Final Tour

Figure 2.1. The Savings Method Illustrated

form a link. Next, the city nearest the last city added is joined to the link. The method continues in this manner until all cities have been included in the tour. Then the first and last cities are joined together. This heuristic requires on the order of n^2 computations (Golden et al., 1980).

Webb (1971) compared the results of the proximity approach with four other sequential heuristics which are based on the idea of a "loss" function. Suppose, during the process of building a tour sequentially, it is desired to link a given unlinked city to its two nearest neighbors (excluding any which have already been linked twice). If the nearest city is not linked to the given city, then the minimum extra distance, or loss, to be paid for not doing so would be at least the extra distance necessary to link it to the third nearest city. In building the tour, links are formed in decreasing sequence of the losses which would be incurred if they are not formed.

The loss function thus described is known as "simple distance loss 1". Problems arise with this loss function in three situations:

1. Cities at both ends of the same chain are among the three nearest neighbors to the given city.
2. Cities at both ends of the same chain have the same unlinked city or cities occurring in the two cities nearest them (converse of situation 1).
3. Either or both cities at the end(s) of a chain occur in the two nearest cities to the cities at both ends of a chain.

To overcome these problems, Webb developed a "simple distance loss 2" function and included a FORTRAN routine for its use, but did not give details of its logic. In a series of 500-city problems, this loss 2 function consistently outperformed the loss 1 function. Using a simpler form of the loss 2 function, which does not require losses to be recalculated after links are formed, problems up to 2500 cities were solved in an average cpu time of 49 seconds on a CDC 6600.

Several "insertion" heuristics have been used in tour-building algorithms. The first of these is the "cheapest insertion" heuristic (Karg and Thompson, 1964). This procedure is as follows:

Step 1. Choose 2 cities to form a partial tour of length 2.

Step 2. Find the city k not in the tour, and the cities i and j in the tour, such that $c_{ik} + c_{kj} - c_{ij}$ is minimized. Insert k between i and j.

Step 3. If the tour is complete, end. Otherwise, go to Step 2.

Golden et al. (1980) have shown that this procedure requires on the order of $n^2 lg(n)$ computations.

Another insertion method, the "nearest insertion" heuristic (Rosenkrantz et al., 1974) proceeds as follows:

Step 1. Begin with a subtour consisting of city i only.

Step 2. Find city k such that c_{ik} is minimal, and link the two cities together.

Step 3. Find city k not in the subtour closest to <u>any</u> city in the subtour.

Step 4. Find the cities i and j in the subtour such that $c_{ik} + c_{kj} - c_{ij}$ is minimal. Insert k between i and j.

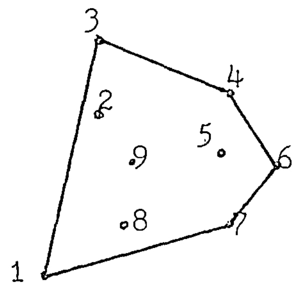
Step 5. If the tour is complete, end. Otherwise, go the Step 3.

Golden et al. (1980) have shown this heuristic to require on the order of n^2 computations.

A "farthest insertion" heuristic (Rosenkrantz et al., 1974) proceeds just as the nearest insertion heuristic above, except that the words "closest to" are replaced by the words "farthest from" in Step 3. This procedure also requires on the order of n^2 computations.

An "arbitrary insertion" heuristic (Rosenkrantz et al., 1974) proceeds as in the nearest insertion heuristic above, except that the city k in Step 3 is chosen arbitrarily. Golden et al. (1980) have shown that this heuristic also requires on the order of n^2 computations.

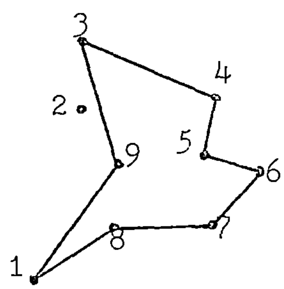
Other insertion tour-building heuristics have been proposed which take advantage of the fact that, for Euclidean problems, the relative ordering of the vertices in the convex hull of the graph remains unchanged in the solution to the TSP (Eilon et al., 1971). Wiorkowski and McElvain (1975) developed a convex-hull procedure which augments an initial extreme-point convex set by iteratively calculating the closest line segments to each free (unassigned) point and assigns new line segments to those points, based on a savings criterion. Refer to Figure 2.2. Norback and Love (1977) inserted new nodes into the convex hull based on the angle formed by the end points of the line segments with the free node at the vertex of the angle. At each step, the free node having the largest angle is inserted between the end points of the line segment, then the angles are recalculated for the next step. For an illustration, see Figure 2.3. Norback and Love (1977) also proposed a method in which new nodes are inserted based on the eccentricity of ellipses formed using the end points of the existing line segments as foci and considering the free nodes to lie on the ellipse thus formed.



(A) Convex Hull (Partial Tour)

Line Segment	Attraction Set
L_{13}	2, 9
L_{34}	∅
L_{46}	5
L_{67}	∅
L_{71}	8

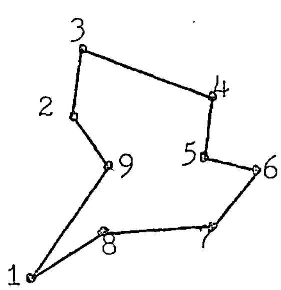
(B) Closest Points to Convex Hull



(C) Partial Tour After Savings Applied

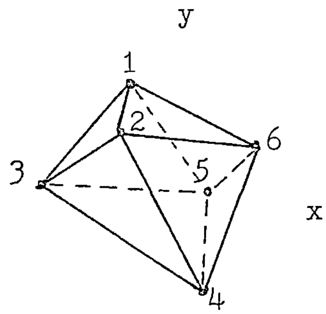
Line Segment	Attraction Set
L_{19}	∅
L_{93}	2
L_{34}	∅
L_{45}	∅
L_{67}	∅
L_{78}	∅
L_{71}	∅

(D) Closest Points to Partial Tour

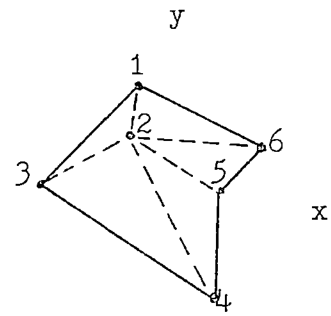


(E) Final TSP Solution

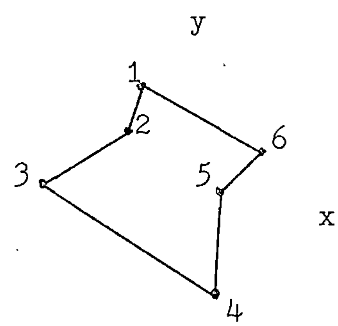
Figure 2.2. Convex-Hull TSP Method of Wiorkowski and McElvain



(A) Angles with vertices at interior points. ∠456 is largest. New partial tour is 1-3-4-5-6-1.



(B) Angles with vertices at remaining interior points. Choose largest one, ∠123 - The tour is 1-2-3-4-5-6-1.



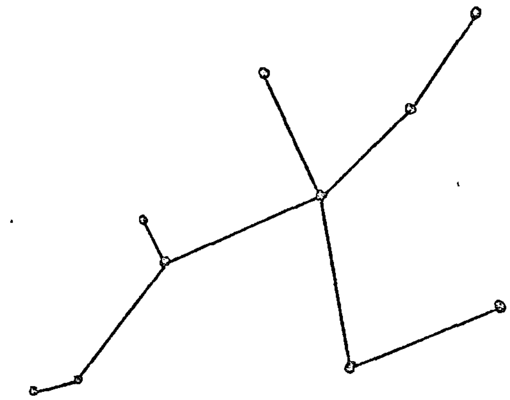
(C) The tour

Figure 2.3. Largest-Angle Convex-Hull TSP Method of Norback and Love
Source: Norback and Love (1977).

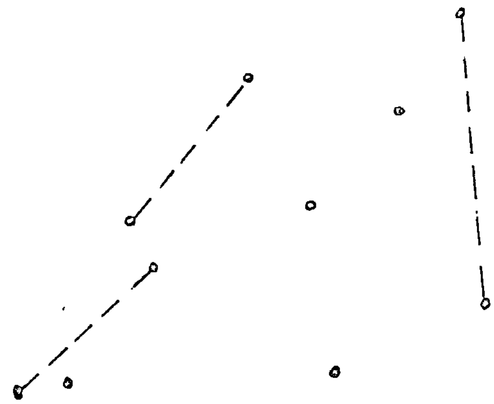
The free node/line segment combination forming the most eccentric ellipse causes the free node to be inserted between the end points. Stewart (1981) presented an algorithm which begins with an initial convex hull and calculates the least cost of inserting each free node k between each pair of line segment end points (i, j). From all (i, j, k) combinations thus obtained, the combination with the minimum ratio $(c_{ik} + c_{kj})/c_{ij}$ is chosen and node k is inserted between i and j. This process is repeated until a complete tour is obtained.

Christofides' heuristic (1979) is a tour-building technique which can be used to solve the TSP using spanning trees and 1-matching. From the minimum spanning tree through the original set of cities, those vertices having odd degree are chosen (if none exist, the problem is solved). Next, the 1-matching problem is solved for this odd-degreed set. The arcs obtained in this matching solution are added to the original arcs in the spanning tree, resulting in a graph containing all even-degreed vertices. Since the vertices are even-degreed, an Euler's tour can be identified through the augmented set of arcs. The Euler's tour is converted to a Hamiltonian circuit by following the Euler's tour (vertex to vertex), adding each vertex in turn to the Hamiltonian circuit. If a vertex is encountered which has already been added to the Hamiltonian circuit, it is skipped. The resulting Hamiltonian circuit is used as the solution to the TSP. Figure 2.4 illustrates Christofides' heuristic.

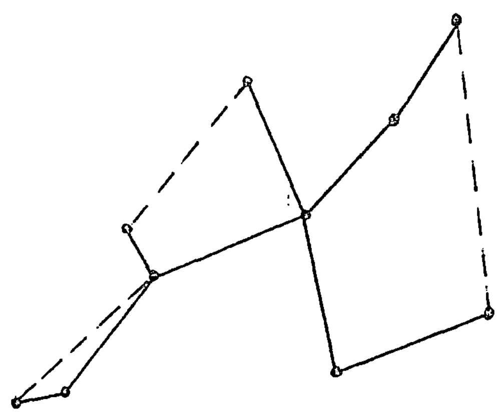
Tour-improvement Heuristics. Given an initial solution to the TSP (from one of the methods of the previous section, for example), the methods of this section attempt to improve that solution through various means. One of the earlier such approaches is due to Croes (1958). For



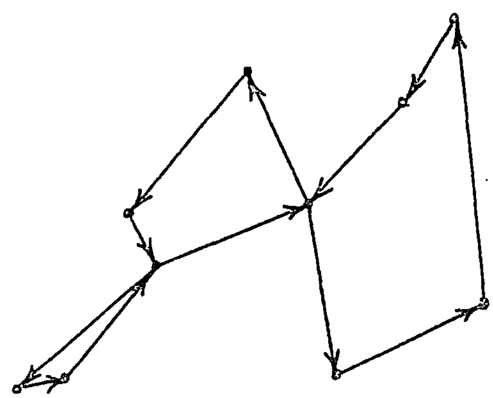
(A) Minimum Spanning Tree



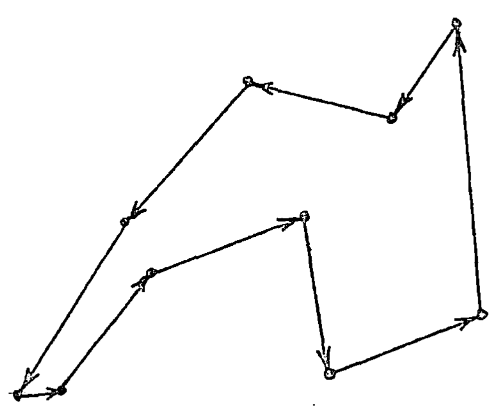
(B) 1-Matching on Odd-Degreed Nodes



(C) Union of Arcs from Minimum Spanning Tree and 1-Matching



(D) Euler's Tour



(E) Hamiltonian Circuit (TSP Solution)

Figure 2.4. Christofides' Heuristic

symmetric problems, it is possible to transform an initial solution through a series of "inversions". These inversions are simple exchanges of arc pairs which yield reductions in the overall travel cost by producing intersectionless routes (routes which do not cross themselves).

Reiter and Sherman (1965) and Lin (1965) independently developed branch-exchange heuristics designed to provide improved solutions to initial tours. The terminology due to Lin will be used here. A tour is said to be k-optimal (or k-opt) if it is impossible to improve the tour by replacing any k of its arcs by any other set of k arcs. By this definition, the intersectionless tours of Croes are seen to be 2-opt. Any tour which is k-opt is also (k-1)-opt. This technique requires on the order of n^k calculations (Golden et al., 1980). Lin and Kernighan (1973) extended the approach to dynamically determine the value of k, based on a set of stopping rules, instead of specifying it in advance.

The original k-opt method as presented by Lin (1965) requires the examination of approximately $\binom{n}{k}(k-1)!2^{k-1}$ combinations of replacement arcs in order to ensure k-optimality (Eilon et al., 1971). This is true because all possible arcs, regardless of their lengths, are examined as possible replacements. However, there are many arc combinations which cannot possibly reduce the overall tour length because of their total length compared with the length of the arcs they are intended to replace. Christofides and Eilon (1972) developed a k-opt procedure which eliminates from consideration all such unrewarding combinations. In a 100-city problem, for example, the number of combinations which were examined to prove 3-optimality was only 18,000 as compared with approximately one million for the original 3-opt method. This advantage becomes even more significant for larger problems.

Stewart (1985) has taken a different approach toward reducing the computational effort required in achieving 3-optimality. This approach requires the calculation of J minimum spanning trees in the following manner:

Step 1. From the original graph, calculate minimum spanning tree. Remove arcs in solution, forming new graph G. Set I = 1.

Step 2. From G, calculate minimum spanning tree. Remove arcs in solution, forming new graph G. Set I = I+1. If I = J, go to Step 3. Else, repeat Step 2.

Step 3. From all arcs in the J spanning trees, select an initial tour.

Step 4. Apply 3-opt procedure, allowing only those arcs in the J spanning trees to be used in exchanges.

Although the computational effort is significantly reduced using this approach, the following points should be noted: (1) A tour might not be found in Step 3. If not, it is possible to obtain a tour by using some arcs not found in the J spanning trees. (2) 3-optimality is not guaranteed by this method, since some of the arcs excluded by the algorithm could conceivably improve the solution.

<u>Composite Heuristics</u>. Golden et al. (1980) investigated several procedures which they terms "composite" methods. The basic composite procedure consists of the following three steps:

Step 1. Use a tour-building heuristic to obtain an initial tour.

Step 2. Apply a 2-opt procedure to the tour found in Step 1.

Step 3. Apply a 3-opt procedure to the tour found in Step 2.

Several variations to the basic procedure exist. In an experiment consisting of several 100-city problems, Golden et al. (1980) found that the basic composite procedure given above will find a solution within two to three percent of optimality in most cases. To bring this figure down to one to two percent, the basic procedure must be repeated several times. The consistently best procedure was found to be one in which either an arbitrary-insertion or farthest-insertion heuristic was used in Step 1, the entire composite procedure being repeated ten times. Computation times are not reported.

The Vehicle Routing Problem

Definition

The vehicle routing problem (VRP) was first proposed by Dantzig and Ramser (1959) as a "truck dispatching problem." Since that time, many modifications to the problem definition (and solution techniques) have been made. Generally speaking, however, the basic VRP can be defined as the problem of generating an efficient set of routes from a central depot to a number of customers, each having a known demand, without violating vehicle capacity constraints or individual route time-or-distance constraints. The VRP has been surveyed by Pierce (1969), Turner et al. (1974), Mole (1979), Lawrence (1981), and Bodin et al. (1983).

The routing problem is sometimes known as the "delivery" problem. However, it should be clear that demand points can involve either pickups or deliveries (some formulations involve both). Typical applications are newspaper delivery, garbage collection, mail delivery, electric meter reading, milk distribution, industrial gas distribution, school-bus routing, etc.

A problem related to the VRP is the vehicle scheduling problem (VSP). Whereas the VRP is basically a spatial problem without time constraints (other than the possible constraint on route duration), the VSP is both a spatial and temporal problem[1]. An example of temporal constraints would be a set of time windows during which the various deliveries or pickups must be made. This category of problems is not covered in this review, with the exception of certain scheduling methods which contain significant contributions to the routing problem.

Classification

Vehicle routing problems can be classified according to several characteristics. The following taxonomy for vehicle routing and scheduling problems is taken from Bodin and Golden (1981):

A. Time to service a particular node or arc

1. time specified and fixed in advance (pure scheduling problem)
2. time windows (combined vehicle routing and scheduling problem)
3. time unspecified (vehicle routing problem,unless there are precedence relationships, in which case it is a combined vehicle routing and scheduling problem)

B. Number of depots

1. one
2. more than one

C. Size of vehicle fleet

1. one
2. more than one

---

[1]The reader is advised that some authors, particularly in the European journals, use the term "vehicle scheduling" when only spatial constraints are involved. This mixing of terms is not so common in the American journals.

D. Type of fleet available

1. homogeneous case (all vehicles the same)
2. heterogeneous case (not all vehicles the same)

E. Nature of demands

1. deterministic
2. stochastic

F. Location of demands

1. at nodes (not necessarily all)
2. on arcs (not necessarily all)
3. mixed

G. Underlying network

1. undirected
2. directed
3. mixed

H. Vehicle capacity constraints

1. imposed - all the same
2. imposed - not all the same
3. not imposed

I. Maximum vehicle route times

1. imposed - all the same
2. imposed - not all the same
3. not imposed

J. Costs

1. variable or routing costs
2. fixed operating or vehicle acquisition costs

K. Operations

1. pickups only
2. deliveries only
3. mixed

L. Objective

1. minimize routing costs involved
2. minimize sum of fixed and variable costs
3. minimize number of vehicles required

M. Other (problem-dependent) constraints

This taxonomy provides a practical means to classify any routing or scheduling problem. Note that under this system, the classical traveling salesman problem (TSP) would be classified as A3, B1, C1, D1, E1, F1, G1, H3, I3, J1, K1, L1.

The objective function to be minimized can actually be of a hierarchal nature. For instance, instead of minimizing the routing costs, a problem formulation might require the minimization of fleet size (L3) and, subject to this minimum fleet size, the minimization of routing costs (L1). Although the minimization of routing costs alone might result in minimum fleet size, there are instances in which the two objectives are not compatible (see, for instance, Eilon et al., 1971, p. 222).

Solution Techniques

The VRP is closely related to the traveling salesman problem (TSP). In the discussion of the TSP, it was pointed out that the TSP is NP-complete, and that problems of moderate size are difficult to solve by exact procedures. This holds more so for the VRP, which is also NP-complete, and which is subject to many more constraints than is the TSP. The classification of solution methodologies which follows in this section is due to Bodin et al. (1983). These procedures include (1) exact methods, (2) cluster first - route second, (3) route first - cluster second, (4) savings and insertion, (5) improvement/exchange, (6) mathematical programming-based, and (7) interactive methods. These different solution methodologies are discussed below.

It should be noted that some procedures for the VRP possess features of more than one of the classifications listed above. In those cases the procedures are classified according to their most salient features.

Exact Methods. An early formulation of the VRP as an integer program was given by Balinski and Quandt (1964). Their formulation requires only that all customers be served and that vehicle capacities not be exceeded. If m feasible routes are designated beforehand, and if the constant d_{ij} designates whether customer j is included on route i (1 = yes, 0 = no), the problem is written as

$$\text{Min} \sum_{i=1}^{m} x_i c_i \tag{2.3}$$

$$\text{S. T.} \sum_{i=1}^{m} d_{ij}\, x_i = 1 \qquad \text{for all } j \tag{2.4}$$

$$x_i = 0,\ 1 \qquad \text{for all } i \tag{2.5}$$

where c_i is the cost of delivering to all customers on route i. It can be seen that x_i equals unity if route i is in the solution, and zero otherwise. Balinski and Quandt used Gomory's cutting plane algorithm to solve the 0-1 program.

The formulation given above is not very useful for four major reasons:

1. The number of feasible routes m can be very large for problems of moderate size.
2. Enumerating all possible feasible routes is difficult.
3. To find the cost of supplying customers on the routes, c_i, a traveling salesman problem must be solved for each route.
4. Other constraints cannot be easily appended to the problem.

Foster and Ryan (1976) also used an integer programming formulation of the VRP. In their method, a feasible set of petal-shaped routes was formulated, and the "over-constrained" problem was solved using linear programming. Cutting planes were used to maintain integrality. The problem was then relaxed to a certain extent to expand the feasible region. Although classified as an "exact" procedure because of the LP approach employed, this method does not guarantee an optimal solution; the time required to relax the constraints to allow consideration of all possible routes is generally prohibitive.

Christofides and Eilon (1969) developed an exact solution procedure based on Little's branch-and-bound tour-building algorithm for the TSP. At each step in the tree, a check is made to ensure that none of the VRP constraints are violated. Also, to prevent unnecessary tree search effort, at each point a feasibility check is made to determine whether sufficient vehicle capacity remains. Bounds are calculated using minimum spanning trees. Using this procedure, VRPs containing up to 12 customers were solved (Eilon et al., 1971).

Christofides, Mingozzi and Toth (1981) present three exact VRP algorithms. Two of the algorithms have bounds calculated from minimum k-degree center trees (k-DCT). A k-DCT is a spanning tree with a given center vertex having exactly k degrees. The third algorithm has bounds calculated from q-routes. A q-route is a least-cost path, from the origin to a node i and back to the origin, for which the total load along the path equals q. In each case, the bounds are calculated by a Lagrangean relaxation ascent method. Computational experience with the algorithms indicate that the bounds calculated from q-routes are superior to those calculated from k-DCTs. VRPs containing up to 25 customers have been solved by this method.

Cluster First - Route Second. This type of procedure solves the VRP in two major steps. The first step assigns customers to individual vehicles by some means, and the second step then sequences the visits among each vehicle's set of customers.

Tyagi (1968) presented an early, very simple version of a cluster first - route second algorithm. His method simply added the nearest neighbor to the last location added, and contiued in this manner as long as vehicle capacity constraints allowed. After m routes were formed in this manner, a TSP solution was obtained for each route. According to Golden, Magnanti and Nguyen (1977) the Tyagi algorithm, while computationally attractive, generally yields inferior solutions.

Gillett and Miller (1974) developed a cluster first - route second procedure called the "sweep" algorithm. It derives its name from the manner in which routes are formed by sweeping an imaginary pointer, fixed at the origin, across all the customer locations in a clockwise manner. Customers are added one at a time to a route as long as vehicle capacity is not exceeded. In this way, m routes are formed. A second step in the sweep algorithm considers exchanges between neighboring routes which serve to reduce the overall distance figure. A third step rotates all the locations counterclockwise so that the first location becomes the last, and the whole process is repeated. This continues until each location has served as the first. The best answer from all such rotations is selected. A second procedure called the "backward sweep" algorithm performs the same steps but the rotations are clockwise instead of counterclockwise.

Gillett and Miller found the time to solve a problem to be a function of the number of routes and the number of locations per route.

The time increased linearly with the number of locations if the route sizes remained approximately constant, and the time increased quadratically with the number of locations per route if the total number of locations remained approximately constant. Problems up to 250 locations were solved using this method.

Other cluster first -route second approaches have been developed by Gillett and Johnson (1976), Chapleau et al. (1981), and Evans and Norback (1984).

Route First - Cluster Second. These procedures work in the reverse order to those discussed above. First, an overall TSP is solved over all the locations, then this overall tour is partitioned into feasible subsets. Several authors have presented versions of the route first - cluster second methodology. Among them are Bodin and Berman (1979), Beasley (1983), Ball et al. (1983) and Golden et al. (1984). The method presented below is due to Beasley (1983):

Let d_{ij} represent the distance between any two locations. Using this distance matrix, a grand tour is formed through all locations, excluding the depot. The locations are renumbered as they appear on the grand tour (depot = 0). Now, define c_{ij} to be the cost of supplying customers $(i + 1, i + 2, . . ., j)$ in any order. This cost is the amount it takes to add a new route containing customers $(i + 1, i + 2, . . ., j)$, given that customer i has been served on a route.

Using the cost matrix thus formed, a network problem is formulated and the shortest path between points is found. The number of vehicles required is equal to the number of arcs containing in the shortest path.

Because the depot is not included in Beasley's formulation, the costs c_{ij} are found by solving a small TSP for each set of locations

(0, i + 1, i + 2, . . ., j). His entire procedure is summarized below:

Step 1. Generate grand tour and use a 2-opt procedure to improve it until further improvements cannot be made using a 2-exchange.

Step 2. Calculate matrix (c_{ij}) using a 2-opt procedure to find TSP solution among customers (i+1, i+2, . . ., j). Add vehicle cost to each c_{ij}.

Step 3. Use Floyd's algorithm to calculate the least-cost paths through the directed graph formed by (c_{ij}), thereby partitioning the grand tour.

Step 4. Use a 3-opt procedure to optimize each individual partition.

Savings/Insertion. The savings method of Clarke and Wright (1964) was covered previously for the TSP. To be applicable to the VRP, the method is altered in only one respect: Each time a link is to be formed by joining two routes together, a feasibility check is made to determine whether vehicle capacity or route duration constraints are met. If so, the routes are joined; if not, the savings link is ignored and the search through the savings list is continued.

Many modifications have been made to the original (Clarke and Wright) version of the savings method. For instance, one of the shortcomings of the original method is that links between locations, once formed, are permanent throughout the duration of the procedure. Tillman and Cochran (1968) proposed a method which overcomes this to some extent. In their method, two routes are not joined permanently until a check is made to determine if the savings might be greater if the connection is not made. A comparison is made among the overall savings resulting from initial connection of the routes showing the highest savings, second-highest, third-highest, etc. Tillman and Cochran reported computational results for only one problem; their algorithm achieved a 6.2 percent reduction in distance over the original Clarke and Wright savings algorithm. A similar approach, involving the "suppression" of savings links already in a given solution and the re-solving of the problem without that link, was reported by Holmes and Parker (1976). Beltrami and Bodin (1974) found that improvements could sometimes be made by simply perturbing the ordering within the savings list, by artifically increasing the distance from the depot to one or more locations, then re-solving the problem subject to the original constraints.

Gaskell (1967) observed that the Clarke and Wright savings method tends to produce peripheral routes which sometimes overlap. To overcome this, the following two savings measures were proposed:

1. $\lambda_{ij} = S_{ij}(\overline{d} + d_{0i} - d_{0j} - d_{ij})$ (2.6)

2. $\pi_{ij} = S_{ij} - d_{ij}$ (2.7)

where S_{ij} = Clarke and Wright savings,

d_{ij} = distance between locations i and j,

and $\overline{d}$ = average distance from depot to all locations.

Both of these measures are modifications of the Clarke and Wright savings measure, and tend to give more emphasis to radially-alligned routes.

The savings algorithm of Clarke and Wright is sometimes known as a "concurrent" or " multiple" version of the method, due to the fact that multiple routes are formed at the same time. While having the advantage that evolving routes compete with one another, the concurrent version has the disadvantage of requiring the entire savings file to be maintained in storage at one time. Yellow (1970), Webb (1971), and Mole and Jameson (1976) introduced "sequential" versions of the savings method which do not require the large savings file. Sequential methods are so named because a route, once begun, is continued as long as feasibility conditions are met. In sequential methods, no competition between routes is present.

An insertion procedure developed by Williams (1982) is known as "proximity priority searching." Beginning with the location most distant from the depot, a new link is formed and the closest two feasible nodes are "pseudo-assigned" (i.e., temporarily assigned) to the link. From that point, the method proceeds as follows:

Step 1. Consider the location next most distant from the depot. If this location has been pseudo-assigned to a link end, make the connection permanent, and find the closest feasible location for pseudo-assignment to the new link end. However, if this (next most distant) location has not been pseudo-assigned to an existing link, add the location to the link list to form the beginning of a new link, and pseudo-assign the closest two feasible nodes.

Step 2. If the closest location is the end of another link, the two links are joined together if a feasible route results.

Step 3. Once a link meets restrictions on load and distance, it is considered complete and is not used for further node assignments.

Step 4. If all nodes have not been assigned, go to Step 1.

This proximity priority searching procedure was tested against six other heuristics over eight problems, and performed as well as any of the others, obtaining the optimal solution in five of the eight problems. A significant fact is that a microcomputer (Cromenco Z2H) was used in Williams' research. The longest computing time was two minutes for a 50-node problem.

Savings and insertion procedures have been extended to the multiple-depot problem. If each city in a VRP is first assigned to the nearest of m depots, and then the Clarke and Wright savings is applied directly to those cities, the indicated savings will not be correct. This is because the savings for linking two cities which are close to one terminal and a greater distance from a second terminal would indicate that the cities should be linked to the farthest terminal, which is incorrect. Tillman and Cain (1972) used a "modified distance" and "modified savings" to overcome this problem. The modified distance is calculated by:

$$\bar{d}_i^k = \min_m d_i^m - (d_i^k - \min_m d_i^m) \tag{2.8}$$

and the modified savings is calculated by:

$$\bar{S}_{ij}^k = \bar{d}_i^k + \bar{d}_j^k - d_{ij} \tag{2.9}$$

where d_{ij} = distance between cities i and j,

and d_i^k = distance from depot k to city i.

In addition to the savings measure, Tillman and Cain introduced a penalty factor for <u>not</u> creating a particular link, and used a weighted combination of the savings and penalty measures in determining which locations to link together. Golden et al. (1977) used the Tillman and Cain approach as the basis for a multiple-terminal algorithm which requires fewer computations at each step and which uses fewer storage locations for the savings matrix.

Improvement/Exchange. These methods begin with an initial solution to the VRP and make improvements to that solution by exchanging the relative positions of customers on a route or between routes. Christofides and Eilon (1969) applied the 3-opt method of Lin (1965), which has been covered previously in this chapter in discussing the TSP. The initial tour of Christofides and Eilon included the same number of (artificial) depots as there were vehicles available. Feasibility checks for vehicle capacity and tour length were made at each exchange. The method produced solutions superior to the savings approach, although at a premium in computational effort. Russell (1977) used a similar approach, except he employed the k-opt method of Lin and Kernighan (1973).

Improvement and exchange heuristics have been applied to the multiple depot problem as well. Newton and Thomas (1974) employed an algorithm similar to Lin's 3-opt method in solving school bus scheduling problems. However, their algorithm was applicable to asymmetric VRPs, so the branch exchanges were limited to those which would not alter the direction of travel in the unchanged portion of the route. Wren and Holliday (1972) applied seven different improvement routines to a set of routes which were initially formed by an insertion procedure. These improvement routines allowed the movement of customers within a route or between routes, as well as the consolidation and recombining of two routes into one.

Mathematical-Programming Based. These procedures, instead of employing "rule of thumb" heuristics, employ heuristics which are based on a mathematical programming formulation of the VRP. A good example of this approach is given by Fisher and Jaikumar (1981). If

K = number of vehicles

n = number of customers

b_k = capacity of vehicle k

a_i = demand of customer i

c_{ij} = travel cost (time, distance, or dollars) from i to j

$$y_{ik} = \begin{cases} 1, & \text{if customer } i \text{ is served by vehicle } k \\ 0, & \text{otherwise} \end{cases}$$

$$x_{ijk} = \begin{cases} 1, & \text{if vehicle } k \text{ travels from customer } i \text{ to } j \\ 0, & \text{otherwise} \end{cases}$$

then the VRP is given by

$$\text{Min} \sum_{ijk} c_{ij} x_{ijk} \qquad (2.10)$$

$$\text{S. T.} \sum_i a_i y_{ik} \leq b_k \qquad k = 1, 2, 3, \ldots, K \qquad (2.11)$$

$$\sum_k y_{ik} = \begin{cases} K, & \text{if } i = 0 \\ 1, & \text{if } i = 1, 2, 3, \ldots, n \end{cases} \qquad (2.12)$$

$$y_{ik} = 0, 1 \qquad i = 0, 1, 2, 3, \ldots, n \quad k = 1, 2, 3, \ldots, K \qquad (2.13)$$

$$\sum_i x_{ijk} = y_{jk} \qquad j = 0, 1, 2, 3, \ldots, n \qquad (2.14)$$

$$\sum_j x_{ijk} = y_{ik} \qquad i = 0, 1, 2, 3, \ldots, n \qquad (2.15)$$

$$\sum_{ij \in S \times S} x_{ijk} \leq |S| - 1 \qquad S \subseteq \{1, 2, 3, \ldots, n\} \quad 2 \leq |S| \leq n-1 \qquad (2.16)$$

$$x_{ijk} = 0, 1 \qquad i = 0, 1, 2, 3, \ldots, n \quad (2.17)$$
$$j = 0, 1, 2, 3, \ldots, n$$

Here, it can be seen that constraints (2.11) through (2.13) apply to a generalized assignment problem, and constraints (2.14) through (2.16) apply to a traveling salesman problem over the customers assigned (by the generalized assignment problem) to a given vehicle k. In reformulating the VRP, Fisher and Jaikumar replace the objective above with the function

$$\text{Min} \sum_{k} f(y_k) \qquad (2.18)$$

where $f(y_k)$ is the cost of an optimal TSP tour of the customers $\{i \mid y_{ik} = 1\}$. This function is defined mathematically as

$$f(y_k) = \min \sum_{ijk} c_{ij}\, x_{ijk} \qquad (2.19)$$

subject to the TSP constraints given above. Since $f(y_k)$ is a very complicated function, a linear approximation is used, instead:

$\sum_{i=1}^{n} d_{ik}\, y_{ik}$. Then the objective becomes

$$\text{Min} \sum_{k} \sum_{i=1}^{n} d_{ik}\, y_{ik}. \qquad (2.20)$$

Solving the generalized assignment problem determines a set of customers for each vehicle. A tour is then constructed through each set using any TSP procedure (Fisher and Jaikumar use Miliotis' exact method, but a heuristic TSP method should also work well).

The values of d_{ik} are determined by first assigning a "seed" customer i_k to each vehicle 1, 2, 3, . . ., K. Then d_{ik} is the cost of inserting customer i into the route from the depot to the seed customer i_k.

$$d_{ik} = \min [c_{0i} + c_{ii_k} + c_{i_k 0}, c_{0i_k} + c_{i_k i} + c_{i0}] - [c_{0i_k} + c_{i_k 0}] \quad (2.21)$$

Cheshire et al. (1982) use a dual heuristic to create a set of routes. In their method, constraints pertaining to vehicle load limits and route time limits are allowed to be violated in the construction of the routes, but at a penalty p•k, where p is a constant expressed in distance units and k is a Lagrangean multiplier. In calculating the cost of inserting a customer into a route, the extra distance required by the insertion and the penalty p•k are added together. At any step, the lowest-cost customer is inserted into the appropriate position. After all customers have been included in the routes, any constraint violations are reduced by iteratively increasing the value(s) of the Lagrangean multiplier(s). In this manner, the procedure maintains local optimality while approaching feasibility.

Stewart and Golden (1984) use a similar approach in their LR3OPT algorithm. Their formulation for the VRP with m vehicles, each having capacity Q, is:

$$\text{Min} \sum_k \sum_{ij} c_{ij} x_{ijk} + \sum_k \lambda_k \left(\sum_{ij} \mu_i x_{ijk} - Q \right) \quad (2.22)$$

$$\text{S. T.} \sum_{ij} \mu_i x_{ijk} - \sum_{ij} \mu_i x_{ijl} \geq 0, \quad l = k + 1 \quad (2.23)$$

$$k = 1, 2, 3, \ldots, m-1$$

$$\lambda_k \geq 0, \quad k = 1, 2, 3, \ldots, m \quad (2.24)$$

where μ_i is the demand at location i. The second half of the objective function is the result of moving capacity constraints, via Lagrangean relaxation, out of the constraint set. The remaining constraint set is simply a method of numbering the routes so that the route which exceeds vehicle capacity by the greatest amount is numbered 1, the route which exceeds vehicle capacity by the next greater amount is numbered 2, etc. The λs are Lagrangean multipliers. In practice, only λ_1 is used in the objective function, which is written as

$$\text{Min} \sum_k \sum_{ij} c_{ij} x_{ijk} + \lambda_1 \sum_{k \epsilon K^+} (\mu_i x_{ijk} - Q) \qquad (2.25)$$

where $K^+ = \{k | \sum_{ij} \mu_i x_{ijk} > Q\}$. (2.26)

The algorithm uses a normal 3-opt exchange procedure to obtain the minimum at each value of λ_1. The values λ_1 are increased geometrically until a feasible solution is found. Published results indicate that the LR3OPT algorithm performs similarly to the dual heuristic of Cheshire et al. (1982).

Interactive. These methods employ a man-machine interface to solve the VRP. Each participant (man and machine) provides input to that part of the problem-solving process for which that participant is best suited. For instance, suitable pairwise changes in a Euclidean routing problem are sometimes obvious to a decision maker when the routes are displayed to him. After the decision maker indicates the pair(s) to be exchanged, the computer can quickly and efficiently calculate the results of the exchange.

There are two types of interaction which can exist between the human decision maker and the computer. First is the case in which the decision maker actually provides part of the solution by making absolute directives to the computer. Moving a customer from one route to another is an example of this type of interaction. "Locking out" a route from the problem is another such example. The second type of interaction is one in which the decision maker guides the progress of problem solution by progressively articulating his preferences to the computer as the solution proceeds. The interactive sequential goal programming procedure of Park (1984) in the solution of multicriteria VRPs contains examples of this type of interaction.

Krolak, Felts and Nelson (1972) developed a two-phase interactive procedure. In the first phase, customers are clustered based on their closeness to each other, without regard to demands, using a standard transportation algorithm. Then these clusters are repeatedly joined pair-wise, based upon the distances between their centers of gravity, until the resulting number of clusters is equal to the prespecified number of routes. The transportation algorithm is again applied to the clusters, resulting in an assignment of vehicles to routes. A TSP routine is applied to each route, then customers are moved from one route to the next until capacity constraints are met. The TSP algorithm is reapplied to the feasible routes, and then simple swapping algorithms are applied in alternation with the feasibility and TSP algorithms until time expires or a specified number of iterations has been performed.

The second phase is the interactive part of the procedure. The decision maker is able to request displays of the current routes, routes in previous solutions, loads and coordinates of customers and vehicles, overloaded vehicles, and current cost. Based upon this information, he may move single customers or strings of customers from one route to another, or exchange a pair of customers in a given route. He may also apply regional heuristics; that is, one or more routes may be isolated for further refinement by one of the VRP heuristics included in the package. Krolak, Felts and Nelson reported that, while the displays provided by Euclidean problems are most comprehensible to the decision maker, other distance metrics can be employed without significantly affecting the ability of the decision maker to recognize possible improvements.

Lawrence (1981) experimented with an interactive procedure which allowed the decision maker to choose from among three different VRP algorithms and three different TSP algorithms in forming and improving a set of routes. A refinement phase then allowed the user to make minor modifications to those routes. Computer graphics were used extensively throughout the interactive package developed by Lawrence.

Scion Consultancy in Great Britain has marketed an interactive VRP package called VANPLAN. Stacey (1983) has reported on a case study involving its implementation. He concluded the following advantages to the interactive method:

1. Non-quantifiable data, such as customer likes and dislikes, can sometimes be traded off against more efficient routes.
2. The system can respond easily to changing parameters.
3. Driver acceptability of the routes is not a concern since an experienced load planner is involved in the route formation.
4. Staff training is less than it would be if algorithmically-planned routes were used.
5. Dependency on a highly skilled staff is reduced.

Stacey also noted the following advantages:

1. Some monitoring of the operators is necessary at first.
2. Management usually has a fear that the best solution will be missed.

Waters (1984) reports on another study involving an interactive routing package. Ten problems were solved and compared, with results obtained by Clarke and Wright (1964), Wren and Holliday (1972), and Foster and Ryan (1976). In general, the total distance obtained by the interactive procedure compared very favorably with that obtained by the other three. Total "hands on" solution times ranged from five minutes for a 6-customer problem to forty-five minutes for a 100-customer problem.

An interactive model used to solve multicriteria VRPs is reported by Park (1984). The VRP under consideration has the following three objectives:

1. Minimize total distance traveled.
2. Minimize deterioration of goods during transport.
3. Maximize fulfillment of priority deliveries and inter-customer precedence requirements.

Park's model uses an "iterative goal programming heuristic" approach. The decision maker can interact with the comuter by progressively articulating his preferences through changes in the priority structure or goal levels, or by directly moving a customer from one route to another. No actual "hands on" solution times are reported for this model.

Workload Balancing in VRPs

Before discussing procedures to balance crew workloads, a short discussion regarding VRP objectives is in order. The original works of Dantzig and Ramser (1958) and Clarke and Wright (1964) regarded the minimization of distance as the VRP's single objective, and many others have followed with the same objective. Other authors (e.g., Gaskell, 1967; Christofides and Eilon, 1969; Foster and Ryan, 1976; and Cheshire et al., 1982) stated their objectives to be hierarchal; i.e. first, the minimization of the number of routes and second, the minimization of both the number of routes and the distance traveled. The fact that some of these formulations involve conflicting objectives received little notice in earlier works. However, Wren and Holliday (1972) observed that

> The cost factors generally considered are the number of trucks, and the total distance traveled. There may be a conflict of objectives here; such conflict has not been resolved in the past. Experience with the current program indicates that no conflict in fact exists (p. 333).

Although Wren and Holliday discounted the conflict between the two objectives, others have considered such conflict. Eilon, Watson-Gandy and Christofides (1971, p. 223) devised an example of such vehicle/distance conflicts. In their example, the total distance of a problem was decreased by adding an extra vehicle to the solution. A similar example occurred in a comparison of five VRP heuristics by Christofides, Mingozzi and Toth (1979, p. 335). This example, which went without comment by the authors, showed a reduction of one vehicle for the Mole and Jameson algorithm over the SWEEP algorithm of Gillett and Miller, but at a distance penalty of slightly greater than four percent.

A question naturally arises regarding the use of single-objective algorithms to solve multiple-objective problems. Just how is the final solution chosen in the face of (non-hierarchal) conflicting objectives? Little insight is provided by the authors of these earlier sources. It can only be assumed that, if different trials of a method produced more than one efficient solution to a given problem, the authors were willing to employ tradeoffs as necessary to choose between them.

Although not part of a particular problem formulation, some authors have recognized that objectives other than distance and/or number of vehicles are important in real-world applications. Golden, Magnanti and Nguyen (1977, p. 125) refer to "unstated goals and/or constraints", and suggest that several solutions be generated by varying a route shape parameter, from which the decision maker can supposedly choose the best solution. Lawrence (1981, p. 102), in detailing the advantages of the interactive man-machine approach to solving the VRP, observed that "unstated objectives and constraints which are difficult to express need not be explicitly stated to the machine." The human decision maker can presumably interact with the machine in such a way that the "unstated objectives" are optimized while the "unstated constraints" are met.

Several authors have alluded to the problem of providing some measure of equity among the tasks assigned to the driver force. For example, Kirby and McDonald (1973) criticized the routes formed by the savings method:

> Routes are often produced which would be quite unacceptable to any transport manager; for example, those which cross themselves, which cross other routes at more than one point, and solutions which include too many short routes. In our opinion, these difficulties call into question the normally accepted criterion of 'optimality' and suggest that there is a place for a subjective factor in assessing optimality (p. 305).

Mole (1979) states that

> In practice, a dispatcher wil also attempt to reconcile the immediate task of providing efficient schedules with a broader view of the business, such as the need to retain equity as between the tasks assigned to several drivers . . . (p. 246).

And finally Waters (1984), in arguing the case for interactive procedures, observes:

> Traditionally, computerized vehicle scheduling has concentrated on minimizing total distance travelled and has largely ignored other objectives, such as minimization of fleet size, variable costs, elapsed time or total travelling time. Other factors for

> consideration might include an equitable distribution of workload amongst drivers and vehicles . . . (p. 821).

Evans and Norback (1984) have developed an algorithm to solve the "time sensitive vehicle routing problem". One of the goals of their method is to obtain routes which satisfy certain restrictions on total route time and driver workloads. For instance, a particular application might require that at least half of the routes contain between eight and ten hours of driver work time. Through the use of a "time density function", customers are first clustered and then routes are formed through the clusters. The time density function is such that heavy insertion penalties are associated with customers near the depot, so these customers are usually assigned to the last route formed. If this last route contains fewer than a pre-specified number of customers, the method attempts to redistribute these customers among the other routes. All of these customers, being near the depot, are also relatively close to each of the other routes. This makes the chances good that these redistributed customers can be used to balance the driver workloads while adding minimal distance to the routes.

Dileepan (1984), in solving the "delivery planning problem", observes that the long-term cost of delivery to a fixed set of customers whose demands vary from period to period is made up to two components: (1) the routing costs within a given period and (2) the cost of changing the routes between periods to ensure feasibility. The number of such route changes can be affected by two factors: (1) the variability of demand from period to period, and (2) the degree to which the routes are balanced. In Dileepan's work, a measure of imbalance is given by the sum of squared deviations in workload (total route time). The objective function to be minimized is a convex combination of the total delivery times and the workload imbalance. To solve the problem

$$\text{Min } \lambda \cdot \text{ total delivery time} + (1 - \lambda) \cdot \text{workload imbalance},$$

eleven different values of λ (0, .1, .2, . . ., 1.0) are selected, and each such problem is solved heuristically using a branch-exchange procedure. In this way, eleven different efficient solution points are generated. The final selection of the best solution is obtained via a computer simulation over a complete planning period which indicates the total impact (delivery cost plus number of changes) over the planning period. Of course, there is no guarantee that all efficient points have been generated by the procedure.

The generation of <u>each</u> efficient point for a 100-customer problem using Dileepan's method took one minute of cpu time on a NAS9000 computer. By extrapolation, the generation of all eleven points of the efficient set should require approximately eleven minutes cpu time. If the number of objectives were increased to three (instead of the two in Dileepan's algorithm), and if the convex multiplier λ is still increased in increments of 0.1 as before, then a minimum of one hour cpu time could be expected to generate the 66 efficient points of the problem.

Husban (1985) employs a "route first -cluster second" arc-routing heuristic in solving the Balanced Tractor-Trailer Routing Problem (BTTRP). In this type of problem, demand is expressed in "move orders" between pickup and delivery points. Because only full trailer loads are transported between points, vehicle capacity is not a concern. To balance the distance traveled by a trailer, Husban uses a minimax criterion, minimization of the longest route. This criterion does not include total time or distance, so a combination of minimum time or distance and the balancing criterion is used as an objective function. The weighting factor used in this combined function is assumed to be known.

Benton (1986) uses a sequential version of the Clarke and Wright savings algorithm to develop initial route assignments, then optimizes each route thus formed by use of Little's tour-building algorithm. Benton claims that this technique results in route sets which are balanced in terms of driving times; however, his comparisons with the original Clarke and Wright algorithm over sixty distance matrices do not include values of an imbalance measure. Mean travel time is used, instead.

Summary

This chapter has reviewed the various types of procedures which have been used to solve the vehicle routing problem, beginning with a review of the closely related traveling salesman problem. Both exact and heuristic techniques were covered for each of these problems.

Exact procedures for the TSP include dynamic programming, linear and integer programming, and branch-and-bound approaches. Heuristic procedures include tour-building, tour-improvement, and composite methods. VRP exact procedures include integer programming and branch-and-bound methods, including the use of Lagrangean relaxation techniques to compute tight lower bounds. VRP heuristic procedures include cluster first - route second, route first - cluster second, savings/insertion, improvement/exchange, mathematical programming-based, and interactive methods.

Earlier approaches to solving the VRP used single-objective procedures, even if the existence of conflicting objectives was recognized. These conflicting objectives were sometimes placed in a hierarchal ranking, and multiple solutions could be compared using this hierarchy, although the single-objective algorithms were not designed to optimize more than one objective. For instance, if the hierarchy of objectives was minimization of the number of vehicles first and minimization of distance traveled second, then multiple solutions from the algorithm would be evaluated by choosing those having the minimum number of vehicles and, within that set, those with the minimum distance traveled. The algorithm, however, "aimed" at only a single (minimum-distance) objective. A notable exception is the algorithm of Park (1984), which uses a heuristic goal programming approach to consider more than a single objective.

As can be seen by the comparison of VRP algorithms in Table 2.1, workload balancing in VRPs has received little attention in the literature. Of the two workload elements in the WBVRP, only the driving time or distance element has received balancing treatment. To do this, two approaches have been taken: (1) incorporate balancing goals in the form of constraints and solve the VRP using a distance-minimizing algorithm, and (2) form several convex combinations of the delivery-time and imbalance objective functions and minimize each resulting function, then select the preferred solution out of the efficient set thus generated. Neither of these approaches provides for an interactive "learning" environment for the decision maker. In addition, the second approach can lead to the solving of a large number of problems, requiring an excessive amount of computer time.

TABLE 2.1

VRP ALGORITHMS

Solution Strategy	Algorithm	Year	Objectives Single	Objectives Mult.	Route Balancing?	Notes
Exact	Balinski & Quandt	1964	x		No	
"	Christofides & Eilon	1969	x		No	
"	Foster & Ryan	1976	x		No	
"	Christofides et al.	1981	x		No	
Cluster First,	Tyagi	1968	x		No	
Route Second	Gillett & Miller	1974	x		No	
"	Gillett & Johnson	1976	x		No	
"	Evans & Norback	1984		x	Yes	1
Route First,	Bodin & Berman	1979	x		No	
Cluster Second	Beasley	1983	x		No	
"	Ball et al.	1983	x		No	
"	Golden et al.	1984	x		No	
"	Husban	1985		x	Yes	2
Savings/Insertion	Dantzig & Ramser	1959	x		No	
"	Clarke & Wright	1964	x		No	
"	Gaskell	1967	x		No	
"	Tillman & Cochran	1968	x		No	
"	Yellow	1970	x		No	
"	Webb	1971	x		No	
"	Tillman & Cain	1972	x		No	
"	Beltrami & Bodin	1974	x		No	
"	Holmes & Parker	1976	x		No	
"	Mole & Jameson	1976	x		No	
"	Golden et al.	1977	x		No	
"	Williams	1982	x		No	
"	Benton	1986		x	Yes	3
Improvement/Exch.	Christofides & Eilon	1969	x		No	
"	Wren & Holliday	1972	x		No	
"	Newton & Thomas	1974	x		No	
"	Russell	1977	x		No	
"	Dileepan	1984		x	Yes	4
Math Prog. Based	Fisher & Jaikumar	1981	x		No	
"	Cheshire et al.	1982	x		No	
"	Stewart & Golden	1984	x		No	
Interactive	Krolak et al.	1972	x		No	
"	Lawrence	1981	x		No	
"	Stacey	1983	x		No	
"	Waters	1984		x	No	5
"	Park	1984		x	No	

[1]Driving times balanced through constraints.

[2]Balanced Tractor-Trailor Routing Problem.

[3]Driving times balanced by sequential savings.

[4]Convex combination of objectives. Near-efficient set generated.

[5]Multiple objectives not specifically stated, but supposedly known by scheduler.

CHAPTER III

GENERAL MODEL STRUCTURE

In this chapter, a general approach is developed to solve the WBVRP. The problem is viewed in the context of multicriteria optimization, and a model structure suitable for its solution in this context is presented. The actual heuristics used to solve the problem are not presented here, but are covered in the next chapter.

Assumptions

In order to begin model development, certain assumptions are necessary. The assumptions which have been adopted for this research are:

1. The vehicle fleet is homogeneous (equal capacity), or the route planner is willing to accept the least capacity in the fleet as a constraint for all vehicles in the fleet.
2. Demand at each customer location is known and deterministic.
3. The distance matrix is symmetric, but not necessarily Euclidean. If non-Euclidean distances are to be used, it is assumed that the distance between each pair of customer locations has been determined by a distance-minimizing procedure.
4. Physical route limitations such as one-way streets, traffic

congestion, road conditions, detours, etc. are ignored.

5. Individual routes are constrained by vehicle capacity, and may or may not be constrained by maximum distance, depending on the problem chracteristics.
6. No explicit utility function is assumed. Instead, an interactive learning environment for the route planner is desired.
7. Satisficing solutions are acceptable. This implies the acceptability of non-optimal solutions.
8. For workload balancing purposes, a crew is assigned to only one route. This precludes the grouping together of two or more short routes in order to balance the driving distances.
9. User interaction with the model may be of two types:
   a. Preemptive
   b. Preference articulation

Mathematical Representation of the WBVRP

The WBVRP is now formulated as an integer program having three objective functions. Assume there are K vehicles, each having a given capacity. Define the problem as

$$\text{Min} \quad f_1 = \sum_{k} \sum_{i,j} c_{ij} x_{ijk} \tag{3.1}$$

$$\& \quad f_2 = \sum_{i,j} c_{ij} x_{ije} - \sum_{i,j} c_{ij} x_{ijf} \tag{3.2}$$

$$\& \quad f_3 = \sum_{i,j} d_j x_{ijg} - \sum_{i,j} d_j x_{ijh} \tag{3.3}$$

S. T. $x_{ijk} \in S_K$ $\forall$ ijk (3.4)

$x_{ijk} = 0, 1$ $\forall$ ijk (3.5)

$\sum_{i,j} d_j x_{ijk} \leq C$ $k = 1, 2, 3, \ldots, K$ (3.6)

$\sum_{i,j} c_{ij} x_{ijk} \leq D$ $k = 1, 2, 3, \ldots, K$ (3.7)

$\sum_{i,j} d_j x_{ijk} - \sum_{i,j} d_j x_{ijl} \geq 0$ for $l = k+1$ & $k = 1, 2, 3, \ldots, K-1$ (3.8)

$\sum_{i,j} c_{ij} x_{ijm} - \sum_{i,j} c_{ij} x_{ijn} \geq 0$ for $n = m+1$ & $m = 1, 2, 3, \ldots, K-1$ (3.9)

where f_1 = total distance

f_2 = route length deviation

f_3 = route load deviation

c_{ijk} = distance traveled between i and j by vehicle k

x_{ijk} = binary variable indicating whether i and j are served by vehicle k (yes = 1, no = 0)

d_j = demand at location j

C = vehicle capacity

D = route distance limit

S_K = set of feasible solutions to the K-TSP

e = vehicle serving route with greatest distance

f = vehicle serving route with least distance

g = vehicle serving route with greatest demand

h = vehicle serving route with least demand

Equations (3.1) through (3.9) define a vector-maximum version of the VRP having K vehicles. Objective function (3.1) calls for the minimization of total distance, (3.2) calls for the minimization of route length deviation, and (3.3) calls for the minimization of route load deviation. Constraint set (3.6) defines the load limit of each vehicle, and constraints (3.7) define the maximum length of each route. Equations (3.8) and (3.9) serve as a means of numbering the routes from the longest to shortest and from most heavily loaded to least heavily loaded, respectively (much in the same manner as the model of Stewart and Golden, 1984).

Multiple Criteria Optimization

The WBVRP, because it contains three different objectives (minimization of total distance, route load deviation, and route length deviation), is a multicriteria optimization problem. The purpose of this section is to provide a brief overview of the area of optimization using multiple criteria.

Terminology

Several terms are used in describing multiple criteria optimization. The following definitions are given by Zeleny (1982):

1. Attributes - descriptors of objective reality. This term refers to traits which can be measured, either objectively or subjectively. In the WBVRP, the attributes are total distance and route-length deviation measures (miles, kilometers, etc.) and measures of route-load deviation (pounds, gallons, number of passengers, number of customers, etc.).
2. Objectives - directions of improvement or preference along individual attributes or complexes of attributes. The three objectives in the WBVRP are in the direction of minimization.
3. Goals - particular target levels of achievement which can be defined in terms of both attributes and objectives. A multiple

criteria optimization problem may or may not include goal values, depending upon the solution methodology employed.

4. Criteria - all the attributes, objectives, or goals which have been judged relevant in a given decision situation by a particular decision maker.

Methods

All multiple criteria optimization problems must contain a set of quantifiable objectives, a constraint set, and a means of obtaining tradeoff information (explicit or implicit) between the objectives. Hwang and Masud (1979) provide a taxonomy of multiple criteria decision making based upon the stage in the decision process in which the tradeoffs are obtained via the decision maker's articulation of preference information. The following discussion follows their taxonomy.

No Preference Information Given. In this category, the decision maker provides no tradeoff information at all, accepting instead the solution provided by the method without qualification. The principal method in this category is the method of global criterion (Boychuk and Ovchinnikov, 1973) in which a global criterion, such as the deviation sum of squares from the feasible ideal points, is minimized. The major disadvantage of this type of procedure is that solutions which are totally unacceptable to the decision maker may be generated.

'A Priori' Preference Articulation. Methods in this category require the decision maker to supply preference information prior to solving the problem. Utility function methods (Keeny, 1972), (Fishburn, 1974), and (Farquhar, 1977) convert the problem to

$$\text{Max} \quad U(f_1, f_2, \ldots, f_k) = U(\underline{f}) \tag{3.10}$$

$$\text{S.T.} \quad g_j(x) \leqslant 0 \qquad j = 1, 2, 3, \ldots, m \tag{3.11}$$

where $U(\underline{f})$ is the overall utility function of the K multiple objectives. The problem is then solved using any suitable single-objective optimization technique. The difficulty with utility function methods is that the determination of $U(\underline{f})$, even for small problems, is not an easy task.

Bounded objective methods (Hwang and Masud, 1979) require the decision maker to supply a minimum acceptable level of achievement for each objective function. The problem is converted to

$$\text{Max} \quad f_r(x) \tag{3.12}$$

$$\text{S. T.} \quad g_i(x) \leqslant 0 \qquad i = 1, 2, 3, \ldots, m \tag{3.13}$$

$$f_j(x) \geqslant L_j \qquad j = 1, 2, 3, \ldots, K \tag{3.14}$$

where L_j is the minimum acceptable achievement level of the jth objective function. Since the decision maker must supply values of L_j in an information void, the method is difficult to apply. In such information voids, the creation of inconsistent constraint sets is possible. Also, it is not always apparent which objective function should be used for $f_r(x)$. Bounded objective methods are rarely used alone, although they may be used as part of other methods (e.g., Benson, 1975).

Goal programming methods have been developed by Charnes and Cooper (1961), Lee (1972), Ignizio (1976), and others. These methods require the decision maker to set goals beforehand for each of the objectives in the problem. The optimal solution to the problem is one in which the deviations from these goals are minimized. The lexicographic version of the method requires the decision maker to also supply an ordinal ranking of the objectives. The problem in K objectives is expressed as

$$\text{Min} \quad P_1h_1(\underline{d}^-,\underline{d}^+),\ P_2h_2(\underline{d}^-,\underline{d}^+),\ \ldots,\ P_lh_l(\underline{d}^-,\underline{d}^+) \tag{3.15}$$

$$\text{S. T.} \quad g_j(x) \leq 0 \qquad j = 1, 2, 3, \ldots, m \tag{3.16}$$

$$f_i(x) + d_i^- - d_i^+ = b_i \qquad i = 1, 2, 3, \ldots, K \tag{3.17}$$

$$d_i^-,\ d_i^+ \geq 0 \qquad \forall\ i \tag{3.18}$$

$$d_i^- \circ d_i^+ = 0 \qquad \forall\ i \tag{3.19}$$

The P_i's are preemptive weights; i.e., $P_i >>> P_{i+1}$. This implies that there is no value w which will make $w \cdot P_{i+1} > P_i$. The terms d_i^+ and d_i^- are positive and negative deviations from the ith goal, respectively. The h_i $(\underline{d}^-, \underline{d}^+)$ are linear functions of the deviations and are referred to as achievement functions.

Progressive Preference Articulation. These methods, usually called 'interactive' methods, do not require the decision maker to express any preferences beforehand. Instead, the decision maker must provide only local tradeoff information as the methods proceed from one solution to the next. As the methods progress, the decision maker also learns more about the problem being solved. Hwang and Masud (1979) list the following advantages to this type of procedure:

1. There is no need for preference information beforehand.
2. A learning process is involved.
3. Only local preference information is required.
4. There is a greater chance of implementation, since the decision maker is more actively involved in obtaining the solution.

5. There are less restrictive assumptions compared with 'a priori' preference articulation methods.

Tradeoffs by the decision maker can be either explicit or implicit. Those methods involving explicit tradeoffs require the decision maker to choose between specific achievement levels of the objectives. Included in this class are the method of Geoffrion et al. (1972), Dyer's Interactive Goal Programming (1972), the Surrogate Worth Tradeoff Method (Haimes, Hall, and Freedman, 1975), the Method of Satisfactory Goals (Benson, 1975), and the method of Zionts and Wallenius (1976).

Those methods involving implicit tradeoff information do not require the decision maker to choose between specific achievement levels of the objectives. Instead, only acceptable achievement levels must be indicated. There are two advantages to this. First, the decision maker is usually more confident in expressing those acceptable achievement levels. Second, the decision maker does not have to be concerned with the range of validity of tradeoffs, which is usually quite narrow for the methods requiring explicit tradeoffs. Methods in this class include the Step-method (STEM) of Benayoun et al. (1971), Zeleny's Displaced Ideal (1982), GPSTEM (Fichefet, 1976), and Steur's Interactive MOLP (1977).

'A Posteriori' Preference Articulation. In these methods, a subset of nondominated solutions is generated. Then from this subset, the decision maker must choose the preferred solution. The major disadvantage of these methods is that the set of solutions from which the decision maker must choose is usually very large. For this reason, they are usually incorporated into interactive methods instead of being used alone. Methods in this class include the Parametric Method (Gal and

Nedoma, 1972), the ε-Constraint Method (Haimes, Hall and Freedman, 1975), and MOLP methods of Steur (1973) and Yu and Zeleny (1975).

The Method of Satisfactory Goals

One of the interactive methods mentioned above, and the one chosen as a basis for solving the WBVRP, is the Method of Satisfactory Goals (Benson, 1975). This method requires the decision maker to identify the 'least satisfactory' achievement level of the K objectives at each iteration. This 'least satisfactory' objective function is then optimized, subject to constraints formed by the original problem constraint set and the acceptable (satisfactory) achievement levels of the remaining objectives. The procedure is rather straightforward, consisting of the following four steps:

Step 1. Choose feasible satisfactory levels of all objectives.

Step 2. Choose the least satisfactory achievement level. If none can be identified as least satisfactory, stop. The final solution has been found.

Step 3. Maintaining other satisfactory achievement levels as constraints, optimize the least satisfactory objective.

Step 4. If improvement in the least satisfactory achievement level (from Step 3) is not sufficient, revise one or more of the constraining achievement levels and go to Step 3. Otherwise, the achievement level for the least satisfactory objective may be revised. Go to Step 2.

The starting point in Step 1 is one in which all objective functions have satisfactory achievement levels. This is a solution which is minimally acceptable to the decision maker, but which he/she would improve upon if possible. In Step 2, the decision maker simply decides which achievement level is farthest from a desired level. If none can be chosen, then all achievement levels are equally satisfactory and the procedure ends with the current solution. Step 3 is the optimization step. The first time this step is performed, the solution is driven toward the nondominated solution set (the beginning solution in Step 1 being feasible but not necessarily nondominated). In subsequent iterations, the solution is moved along the nondominated set. In Step 3, the decision maker must decide whether the least satisfactory achievement level has been sufficiently improved. If not, one or more of the constraining achievement levels must be relaxed in order to allow further improvement to take place. The decision maker is helped in this step by having values of dual variables which indicate, on a local level, the consequences of given levels of constraint relaxation.

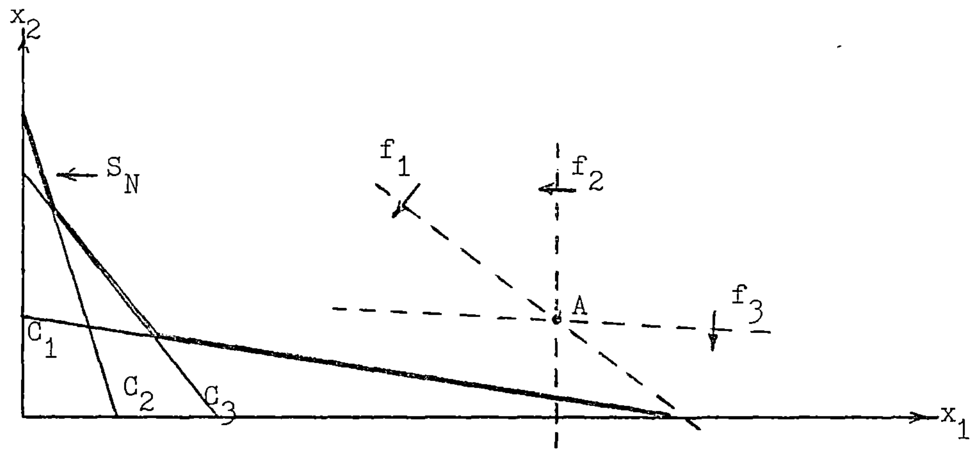
The Method of Satisfactory Goals may be applied to linear or non-linear problems, in continuous or integer variables. Figure 3.1 illustrates the method applied to a linear problem in two variables having three objective functions f_1, f_2, and f_3. Problem constraints are C_1, C_2, and C_3, and the nondominated solution set is shown by the heavy lines S_N. In Figure 3.1.a, the decision maker has selected point A as the initial feasible solution. Note that this solution is not a member of S_N. Suppose the decision maker selects f_3 as the objective function with the least satisfactory achievement level. In this case, f_3 is then minimized while the achievement levels of f_1 and f_2 become (inactive) constraints. This results in solution point B, shown in Figure 3.1.b. This solution is a member of S_N. Now, none of the objective functions can be minimized further unless the decision maker is willing to relax at least one of the other achievement levels. Suppose that the achievement level of objective function f_1 is next chosen as the least satisfactory. Figure 3.1.c shows that the achievement level of objective function f_3 must be relaxed in order for f_1 to be improved. This relaxation results in the feasible subspace defined by the shaded area in Figure 3.1.c. It is obvious that the minimization of f_1 over this feasible subspace will result in solution point C, from which the procedure will continue.

The Method of Satisfactory Goals proceeds in this manner, always optimizing an objective function over a feasible subspace determined by the amount of relaxation applied to the other achievement levels. Because the solution space is a subset of the original feasible space, and because this subspace is bounded in part by specific achievement levels, Benson refers to the solution obtained as 'quasi-efficient'.

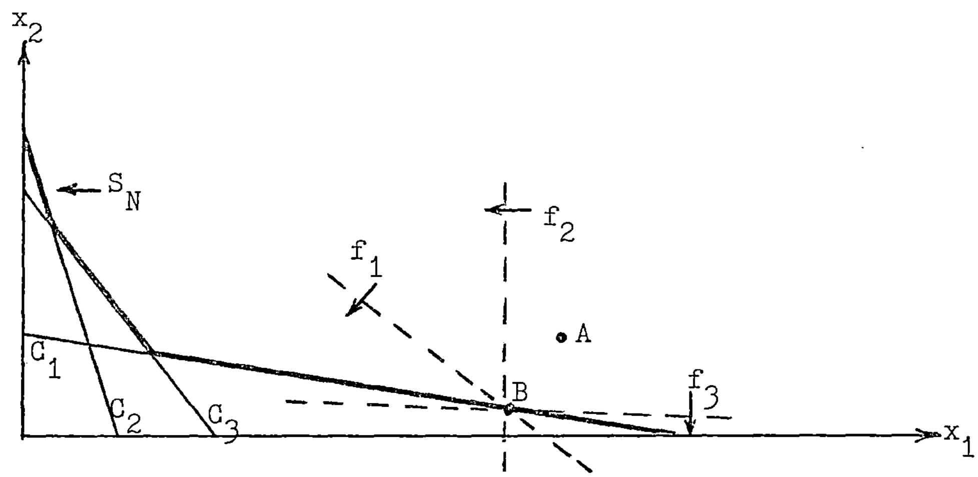
The Method of Satisfactory Goals Applied to the WBVRP

The vector-maximum problem defined by (3.1)-(3.9) could, in theory, be solved as a multiobjective integer program (MOIP). Some work has been done toward solving multiobjective integer programs; articles by Lee (1977), Zionts (1977), Villarreal, Karwan, and Zionts (1980), and Klein and Hannan (1982) are representative of research in this area. However, since the underlying VRP is NP-complete, the difficulty of employing such an exact approach to solve problems of reasonable size should be obvious. Recall that the exact approach of Christofides, Mingozzi, and Toth (1981) was used to solve (single objective) VRPs of no more than twenty-five customers.

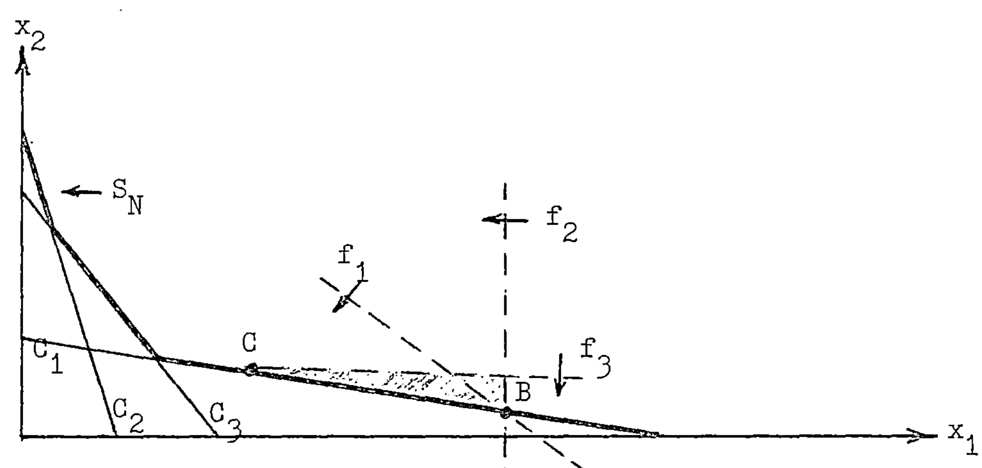
Faced with the almost impossible task of optimally solving the



a. Initial Satisfactory Feasible Solution



b. Minimization of Objective Function f_3



c. Achievement Level of f_3 Relaxed. f_1 to be Minimized Next.

Figure 3.1. Method of Satisfactory Goals Illustrated

multiobjective VRP, it would seem that a reasonable approach would be to use heuristic methods. Gabbani and Magazine (1985) use an interactive heuristic approach to solving the MOIP. Their approach, based on Steur's method of interval criterion weights (1977), requires that 2K + 1 single-objective problems be solved and presented to the decision maker at each iteration of the process, where K is the number of objectives in the MOIP. The decision maker is then asked at each iteration to choose his preferred solution from among the 2K + 1 alternatives. Gabbani and Magazine employ a single objective binary integer program (SOIP) heuristic for each of these 2K + 1 problems at each iteration. In a series of (0,1) problems solved on an IBM 4341, computation times using this heuristic ranged from 11.72 to 15.78 seconds for an MOIP having three objectives, thirty constraints, and sixty variables. Although extrapolation of these times to a (0,1) problem the size of(3.1) - (3.9) (over 300 variables in more than 3000 constraints for a ten-customer, three-vehicle problem) would no doubt be inaccurate, it should nevertheless be obvious that a general-purpose SOIP heuristic cannot be successfully employed. Use of a special-purpose VRP heuristic (e.g., a 3-opt branch exchange heuristic) to solve each of the seven problems at each iteration would reduce the effort some, but overall computation times would likely still be excessive. Examination of this approach would be a good topic for future research, however.

Even if the solution times of the Gabbani and Magazine approach could be reduced through the use of a different SOIP heuristic, the decision maker still has the task of choosing a preferred solution from among seven different sets of routes at each iteration. This amount of user input, plus the requirement that the decision maker be consistent with his preferences, seems somewhat excessive. An acceptable alternative, it would seem, would be the use of an interactive satisficing algorithm in which the user expresses his preference for 'satisfactory' levels of objective function achievement. One such approach, the Method of Satisfactory Goals (discussed above), provides this capability. The three objective functions of the WBVRP were given by equations (3.1), (3.2), and (3.3) as before. If f_1 has been designated by the decision maker as the objective function having the least satisfactory achievement level, and if b_2 and b_3 are the previously attained achievement levels of f_2 and f_3, respectively, then the problem is formulated as

$$\text{Min} \quad f_1 \tag{3.20}$$

$$\text{S. T.} \quad f_2 \leqslant b_2 \tag{3.21}$$

$$f_3 \leqslant b_3 \tag{3.22}$$

and (3.4) - (3.9).

Similarly, if f_2 has been designated as the objective function having the least satisfactory achievement level, the problem is defined as

$$\text{Min} \quad f_2 \tag{3.23}$$

$$\text{S. T.} \quad f_1 \leqslant b_1 \tag{3.24}$$

$$f_3 \leqslant b_3 \tag{3.25}$$

and (3.4) - (3.9).

Finally if f_3 has been designated as the objective function having the least satisfactory achievement level, the problem is defined as

$$\text{Min} \quad f_3 \tag{3.26}$$

S. T. $f_1 \leqslant b_1$ (3.27)

$f_2 \leqslant b_2$ (3.28)

and (3.4) - (3.9).

While the general approach of the Method of Satisfactory Goals can be used to solve the WBVRP, the method cannot be used without modification. Due to the fact that the WBVRP cannot be solved optimally, heuristics must be employed to minimize the given objective function at each iteration of the method. This has three implications for the model:

1. The heuristics which are developed for the three objective functions should be efficient (i.e., capable of solving problems within reasonable computing times) and effective (i.e., capable of providing good, albeit non-optimal, solutions). This issue is covered in Chapter IV.
2. The model should provide the decision maker with guidance concerning the consequences of constraint relaxation at any iteration. The Method of Satisfactory Goals supplies the decision maker with values of dual variables for this purpose. Heuristic procedures cannot do this, so other means of providing guidance to the decision maker must be investigated. This issue is covered in Chapter V.
3. Heuristics, which cannot guarantee optimality in the single-objective case, cannot guarantee nondominance in the multi-objective case. The decision maker must settle for a 'near efficient' solution (Gabbani and Magazine, 1985). This, too, is discussed in Chapter V.

A flowchart depicting the general WBVRP model structure is shown in Figure 3.2. The procedure begins with the determination of an initial satisfactory route set. This route set is determined using a heuristic distance-minimizing algorithm. Route length deviation and route load deviation are unconstrained. It is assumed that any decision maker is willing to consider a minimum-distance solution 'satisfactory', and that further solutions can proceed from that point. If the decision maker is willing to accept this initial solution without change, the procedure halts. Otherwise, the current value of either route length deviation or route load deviation is chosen as the least satisfactory achievement level, LS. Depending upon the choice of LS, a heuristic length-deviation or load-deviation algorithm is employed to obtain a new solution. This corresponds to the initial execution of Step 3 in the Method of Satisfactory Goals, in which a nondominated solution is sought. From that point on, the model proceeds just as in Steps 2 through 4 on page 62. The procedure ends when no achievement level can be designated as least satisfactory.

When any given solution is displayed, there is a possibility that the decision maker can make a preemptive route adjustment to improve one or more of the routes in the solution. Route adjustments are limited to a single route at a time; i.e., only distance-minimizing adjustments are allowed. Route-length and route-load deviation adjustments are not made preemptively.

Summary

In this chapter, a general model structure to solve the WBVRP has been presented. Assumptions necessary for model devlopment were made,

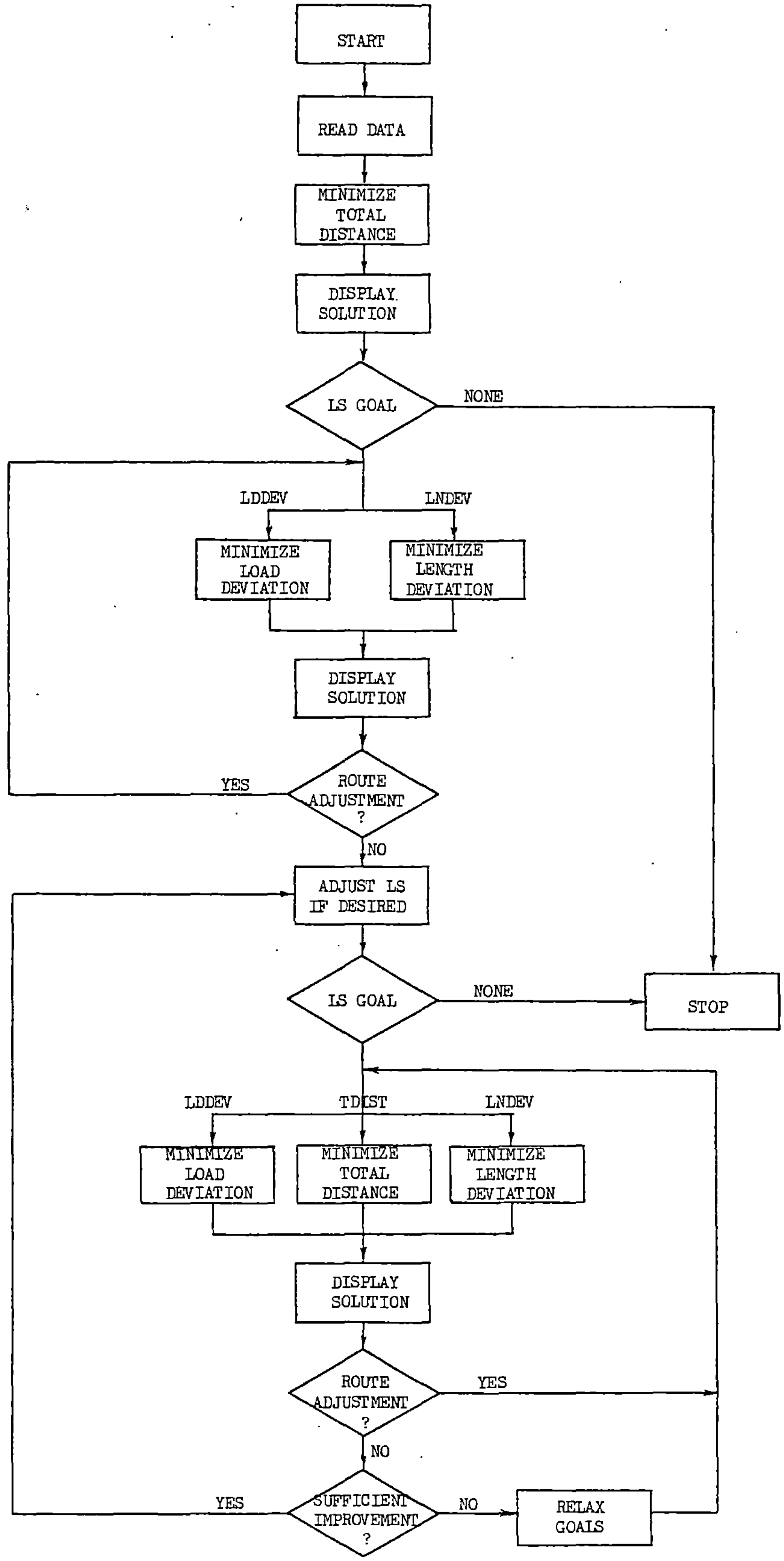


Figure 3.2. General WBVRP Model Structure

and a mathematical formulation of the problem as a vector-maximum (0,1) integer program was given. This was followed by a brief overview of multiple criteria optimization methods. Finally, a heuristic version of one of those methods, the Method of Satisfactory Goals, was chosen as the basis for solving the problem. In this approach, one of three objective functions must be minimized at each iteration, subject to satisfactory achievement levels of the remaining two objective functions. The heuristics employed to minimize these objective functions were not covered here. They are the subject of the next chapter.

CHAPTER IV

SINGLE-OBJECTIVE ALGORITHMS

In the previous chapter, a general model structure for solving the Workload-Balanced Vehicle Routing Problem (WBVRP) was presented. This model involves use of the Method of Satisfactory Goals (Benson, 1975), in which a single objective function is minimized at each iteration of the procedure, subject to satisfactory achievement levels of the other objective functions. The single-objective functions which are to be minimized are:

1. Total Distance,
2. Route Length Deviation, and
3. Route Load Deviation.

Since the WBVRP is NP-complete, the use of heuristics to minimize each of the single-objective functions is necessary. The purpose of this chapter is to develop these heuristic algorithms. For each algorithm, analyses of its effectiveness (ability to produce good solutions) and efficiency (computational speed) are also presented.

Minimization of Total Distance

Minimum-Distance Algorithm

If the total distance is selected by the decision maker as the least satisfactory achievement level, then the total-distance objective function must be minimized, subject to the original problem constraints and the satisfactory achievement levels of the other two objective functions. The algorithm chosen to minimize total distance is the 3-opt arc-exchange heuristic of Lin (1965) adapted to the VRP by Christofides and Eilon (1969). The algorithm was chosen because of its ability to generate near-optimal solutions. In eight test problems, the 3-opt algorithm produced solutions which were on the average within 1.96 percent of the best solution found in the literature. For details see Cheshire et al. (1982) and Stewart and Golden (1984).

Recall that a solution is said to be k-optimal if it is impossible to improve the solution by removing k arcs from the routes and replacing them with k other arcs. Figure 4.1 shows the different types of arc exchanges which are employed in the 2-opt algorithm and in the 3-opt algorithm. Note that the last three types of 3-opt arc exchanges correspond exactly to the single 2-opt exchange. It is for this reason that any solution which is 3-optimal is also said to be 2-optimal.

The original k-opt algorithm developed for the traveling salesman problem was concerned with only a single route, the TSP tour. The VRP can have up to K routes, where K is the number of vehicles in the problem. In order to use k-opt algorithms for the VRP, it is convenient to restructure the K routes into a single tour. This is done through the use of 'artificial depots'. An artificial depot has the same characteristics as the original depot (i.e., the same coordinates and zero demand), and it is inserted between any two routes in the problem. To form a single tour, K - 1 artificial depots are necessary. This is illustrated in Figure 4.2. The dashed lines on either side of the artificial depots in Figure 4.2(B) are used to indicate that the distances are not to scale, the artificial depots having been displaced

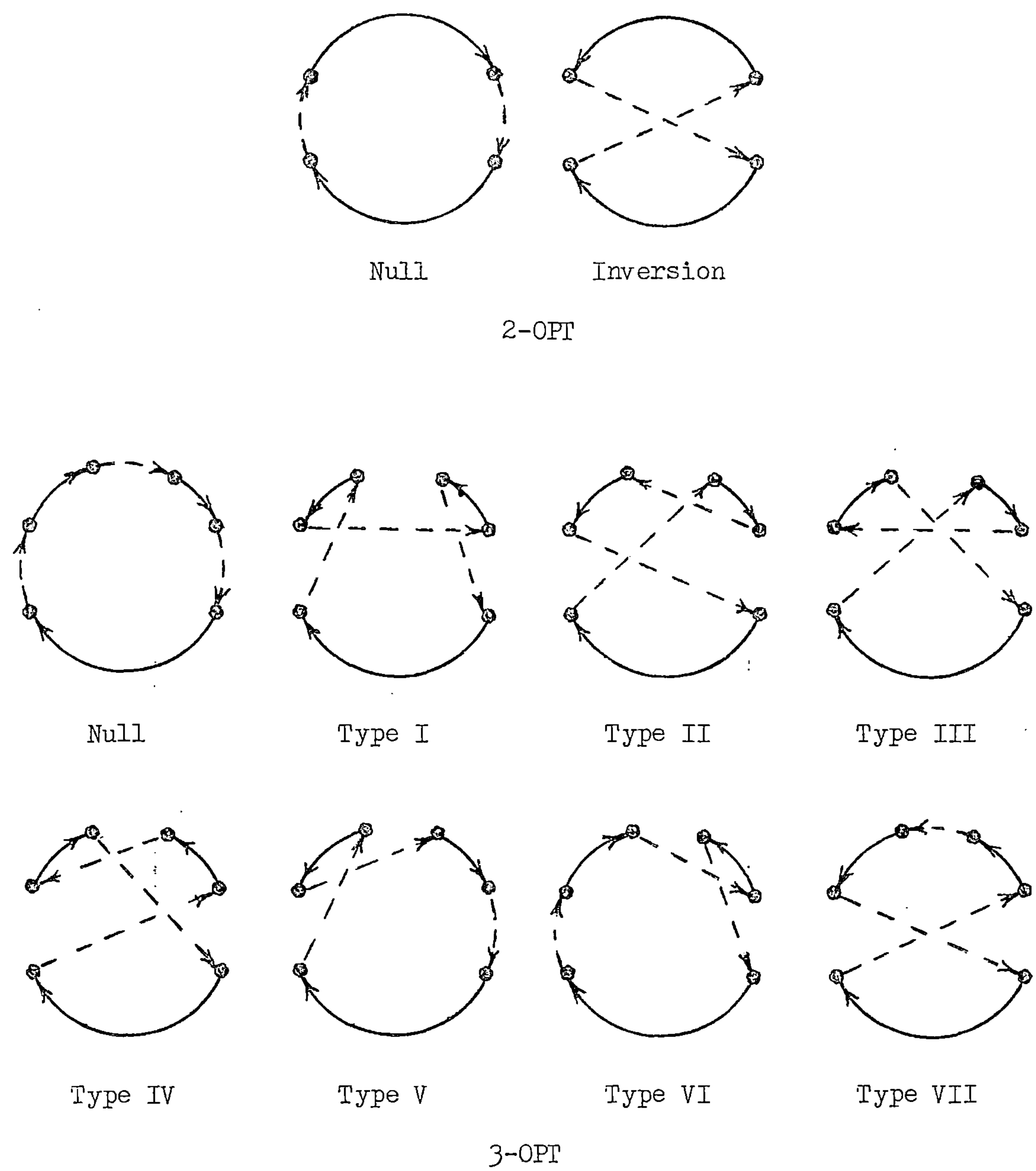
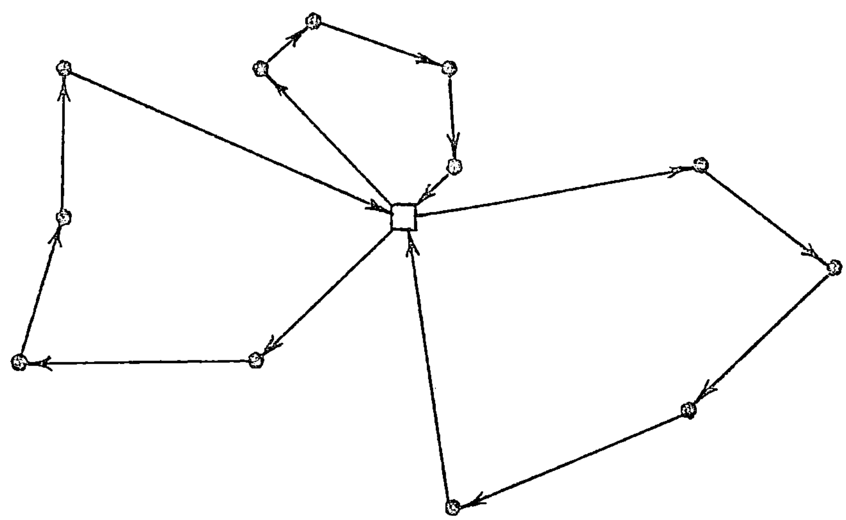
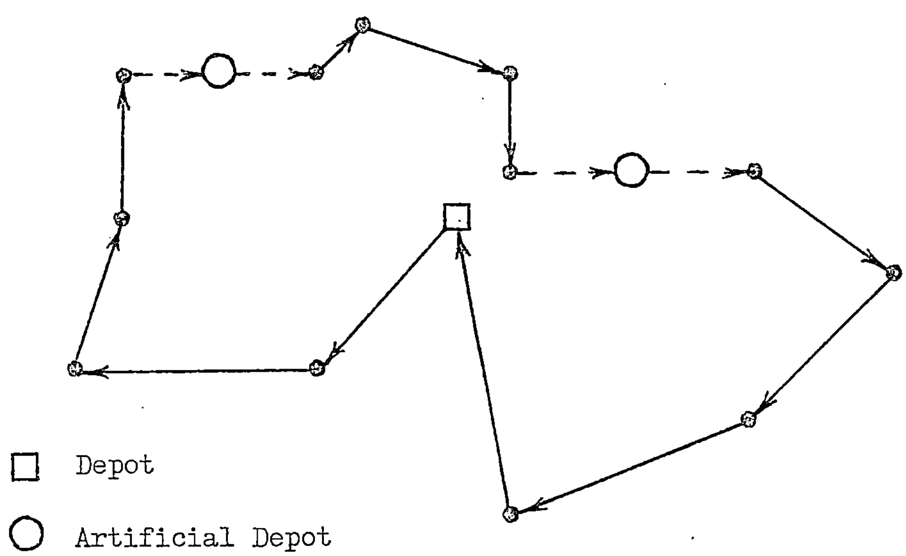


Figure 4.1. 2-opt and 3-opt Arc Exchanges



(A) ORIGINAL ROUTE STRUCTURE



(B) INSERTION OF ARTIFICIAL DEPOTS

Figure 4.2. Restructuring Routes Using Artificial Depots

from the original depot location in the figure. The depots, original and artificial, are treated just as the other nodes in the k-opt arc exchange process. The only difference is that the resulting routes must be checked for vehicle-capacity and route-length feasibility. In addition, for the WBVRP, the route-length deviation and route-load deviation must be examined to determine that they do not exceed the satisfactory achievement levels of those objectives.

Figure 4.3 is a simplified flow chart of the distance minimizing algorithm. It is valid for either a 2-opt procedure (in which case there is only one type of arc exchange) or a 3-opt procedure (in which case there are seven types of arc exchanges). From a practical standpoint, it is usually more efficient to achieve 2-optimality through the 2-opt procedure before submitting the problem to a 3-opt procedure. This is because the heuristics have a computational difficulty which is a function of N^k, where N is the number of nodes in the network (Golden et al., 1980). The k-opt heuristics have been employed in this manner in the current research.

In Figure 4.3, it can be seen that feasibility checks do not have to be made if all of the selected arcs are in the same route. In this case, the route load, route length, and route-load deviation will remain feasible. The only possible infeasibility could be in route-length deviation, and that caused by an improvement in the (supposed) shortest route. Since route-length deviation is <u>never</u> to be minimized through inefficient routing, the arc exchange must take place even if the route-length deviation constraint is violated.

After an arc exchange is made, the route structure is 'rotated'; i.e., the first arc to be examined in the previous search for a valid

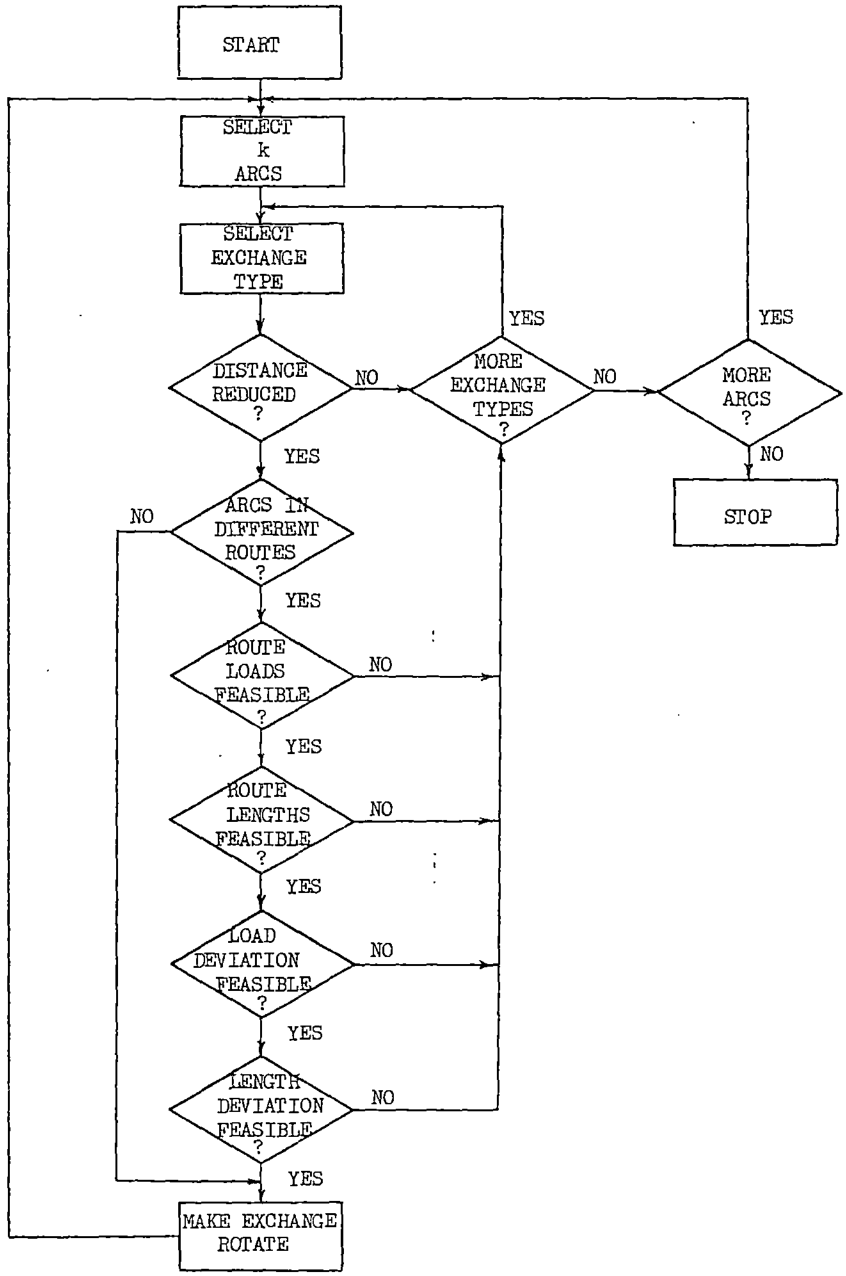


Figure 4.3. Distance Minimizing Algorithm

exchange is replaced in the next search by its predecessor. This is to prevent an exact repetition of the search sequence up to the point of exchange, and increase the likelihood that the next valid exchange will be found quickly. The k-opt procedure ends when all arcs have been examined without finding a valid exchange.

Effectiveness of Algorithm

It is of interest to know how good the answers are which are generated by the distance minimizing algorithm. It was stated above that solutions found by the 3-opt procedure were found to be, on the average, within 1.96 percent of the best solutions published in the literature. However, as is common practice, those 3-opt solutions were found by running the procedure several times (normally ten times) from different starting points and selecting the best one. In an interactive procedure, it is unlikely that this procedure can be followed. It is more likely that restrictions on computing time will limit each distance-minimization subproblem to no more than two or three solutions, from which the best can be selected.

To test the quality of solutions generated by the 3-opt procedure, six problems from the literature and five problems from Chapter VI of this research were selected. The best of one, two, three, and ten runs were selected for each problem and compared with the best known solutions. The best known solutions for the first six problems were taken from the literature, and the best known solutions for the last five problems were found by solving them a minimum of fifteen times each. The solutions and percent errors are shown in Table 4.1. Here, it can be seen that the best of ten runs had an average error of 1.66 percent, which is close to the 1.96 percent found in the literature. Running the algorithm one, two, and three times and selecting the best solution resulted in an average error of 3.75, 3.17, and 3.14 percent, respectively. These figures will be referred to in Chapter V, in which an interactive computer program employing the distance minimizing algorithm is presented.

Efficiency of Algorithm

The algorithms were all programmed in VS FORTRAN and run on the IBM 3081D at Oklahoma State University. Golden et al. (1980) state that the 3-opt algorithm, when applied to the traveling salesman problem, has a computational difficulty on the order of N^3, where N is the number of cities visited. It would seem that the computing times for the distance minimization subproblem of the WBVRP would also be a function of N^3, even though more feasibility checks are necessary in this case (Figure 4.3). To verify this, thirty-six different distance minimization problems were solved using various starting points, and CPU times were determined. The problems selected were Gaskell's 22-city, 29-city, and 32-city problems, and Christofides and Eilon's 50-city, 75-city, and 100-city problems. A model of the form

$$\text{CPU time} = \beta_0 \cdot N^{\beta_1} \qquad (4.1)$$

was chosen, and a regression program was run using the model. Table 4.2 shows the results of the regression. The model obtained from the regression is

$$\text{CPU time (seconds)} = 9.0 \times 10^{-6} \cdot N^{3.1494} \,. \qquad (4.2)$$

TABLE 4.1.

EFFECTIVENESS OF DISTANCE MINIMIZATION ALGORITHM

Problem Number	Source	No. of Cities	Best of n Runs n=1	n=2	n=3	n=10	Best Known Solution	Percent Error n=1	n=2	n=3	n=10
1	Gaskell (1967)	22	958	958	958	949	949	0.95	0.95	0.95	0.00
2	Gaskell (1967)	29	873	873	873	873	873	0.00	0.00	0.00	0.00
3	Gaskell (1967)	32	809	809	809	809	809	0.00	0.00	0.00	0.00
4	Christofides and Eilon (1969)	50	566	566	566	545	521	8.64	8.64	8.64	4.61
5	Christofides and Eilon (1969)	75	858	851	851	851	845	1.54	0.71	0.71	0.71
6	Christofides and Eilon (1969)	100	880	863	860	860	829	6.15	4.10	3.74	3.74
7	Chapter VI	36	479	473	473	455	426	12.44	11.03	11.03	6.81
8	Chapter VI	36	501	501	501	490	490	2.24	2.24	2.24	0.00
9	Chapter VI	36	546	535	535	535	533	2.44	0.38	0.38	0.38
10	Chapter VI	36	627	627	627	627	627	0.00	0.00	0.00	0.00
11	Chapter VI	36	422	422	422	403	395	6.84	6.84	6.84	2.03
Average error								3.75	3.17	3.14	1.66

The parameters of this model are significant at the 0.01 level, and the model's R^2 is 0.97. Estimated CPU times range from 0.15 seconds for a 22-city problem to 17.91 seconds for a 100-city problem, demonstrating the efficiency of the 3-opt algorithm applied to the WBVRP.

Minimization of Route-Length Deviation

Route-Length Deviation Algorithm

If the decision maker chooses route-length deviation as the least satisfactory achievement level, then the route-length deviation objective function must be minimized, subject to the original problem constraints and the satisfactory achievement levels of the other two objective functions (total distance and route-load deviation). The algorithm developed for this subproblem utilizes arc exchanges similar to the k-opt arc exchanges used to minimize total distance. The heuristic used for route-length deviation is different, however, not only in the objective to be minimized, but also in the set of arcs selected for possible exchanges and in the manner in which the validity of an exchange is determined.

Route balancing methods require a means of clustering customers into potential routes, after which each route length is minimized by solving a TSP. Then the route-length deviation is calculated to determine whether an improvement will result. There are different ways of clustering the customers. Dileepan (1984), for example, uses simple pairwise customer exchanges between two routes at a time. In the current method developed for the WBVRP, 2-opt and 3-opt type exchanges are used for clustering, instead. There are four reasons for using these exchanges for the clustering step:

TABLE 4.2

REGRESSION TABLE FOR 3-OPT SOLUTION TIMES

Source	Degrees of Freedom	Sum of Squares	Mean Square	F Value	P(>F)	R^2	Parameter	Estimate	t Value	P(>|t|)
Model	1	102.9525	102.9525	1004.41	0.0001	0.97	$Ln(\beta_0)$	-11.6158	-30.51	0.0001
Error	34	3.4855	0.1025				β_1	3.1494	31.69	0.0001
Total (corrected)	35	106.4380								

1. The means of accounting for the different customer exchanges is already provided in the k-opt logic.
2. Many kinds of trades are examined. Arc exchanges can result in one-for-zero, one-for-one, two-for-zero, two-for-one, two-for-two trades, etc., whereas pairwise customer exchanges result only on one-for-one or one-for-zero trades.
3. Exchanges involving up to three routes at a time can be examined.
4. The relative ordering of customers being transferred from one route to another is preserved in the cluster prior to solving the TSP. This is important, since in many cases this same ordering will be optimal in the new route, also.

The TSP for each newly formed cluster is solved heuristically, since the large number of TSPs necessary in the course of solving the route balancing problem precludes the use of exact methods. In the current research, each TSP is solved twice by a 3-opt procedure using two different starting points. If the number of exchanges evaluated for the route balancing problem were the same as the number of exchanges evaluated for the distance minimizing problem, the solving of TSPs by any means would cause the computation times to be prohibitive. Two points are relevant to this concern:

1. The set of arcs which are candidates for an exchange in the route balancing problem is much smaller than the set of arcs in the distance minimization problem.
2. If two routes are involved in a 3-arc exchange, the number of exchange types in the route balancing heuristic is only three, whereas there are seven exchange types in the 3-opt distance minimization heuristic.

The first point is illustrated by considering a typical route balancing problem. In order to improve the measure of imbalance, more than one route must be involved in the arc exchange. All exchanges involving a single route are ignored. In addition, an exchange must include the longest route, the shortest route, or both; otherwise, no improvement in the minimax criterion could be made.

The second point is illustrated by examining Figure 4.4, which shows the clusters which result from a 3-arc exchange involving two routes. Note that arc (2-3), one of the arcs to be eliminated, is in a route to itself. Here it can be seen that the clusters formed by the Type III exchange are the same as those formed by the Type IV exchange. Similarly, the Type I and Type V exchanges result in identical clusters, as do the Type II and Type VII exchanges. Finally, the Type VI exchange is null, since it forms no new clusters. Therefore, only three of the seven exchange types are necessary to provide the route clustering. Exchange types III, V, and VII are selected, since they preserve the relative ordering of customers in the clusters. Similar conclusions would be drawn if are (5-6) or arc (7-8) had been in a route to itself, although the three exchange types selected would have been different.

Figure 4.5 shows the clusters resulting from a 3-arc exchange involving three different routes. Here, it can been seen that the clusters formed by each type of exchange are different. Therefore, no exchange types can be eliminated in this case.

It should be noted that the 2-arc and 3-arc exchanges do not provide all possible clusterings of customers. All the exchanges except the Type III and Type IV 3-arc exchanges in Figure 4.4 trade contiguous groups of customers beginning with the customer nearest the depot.

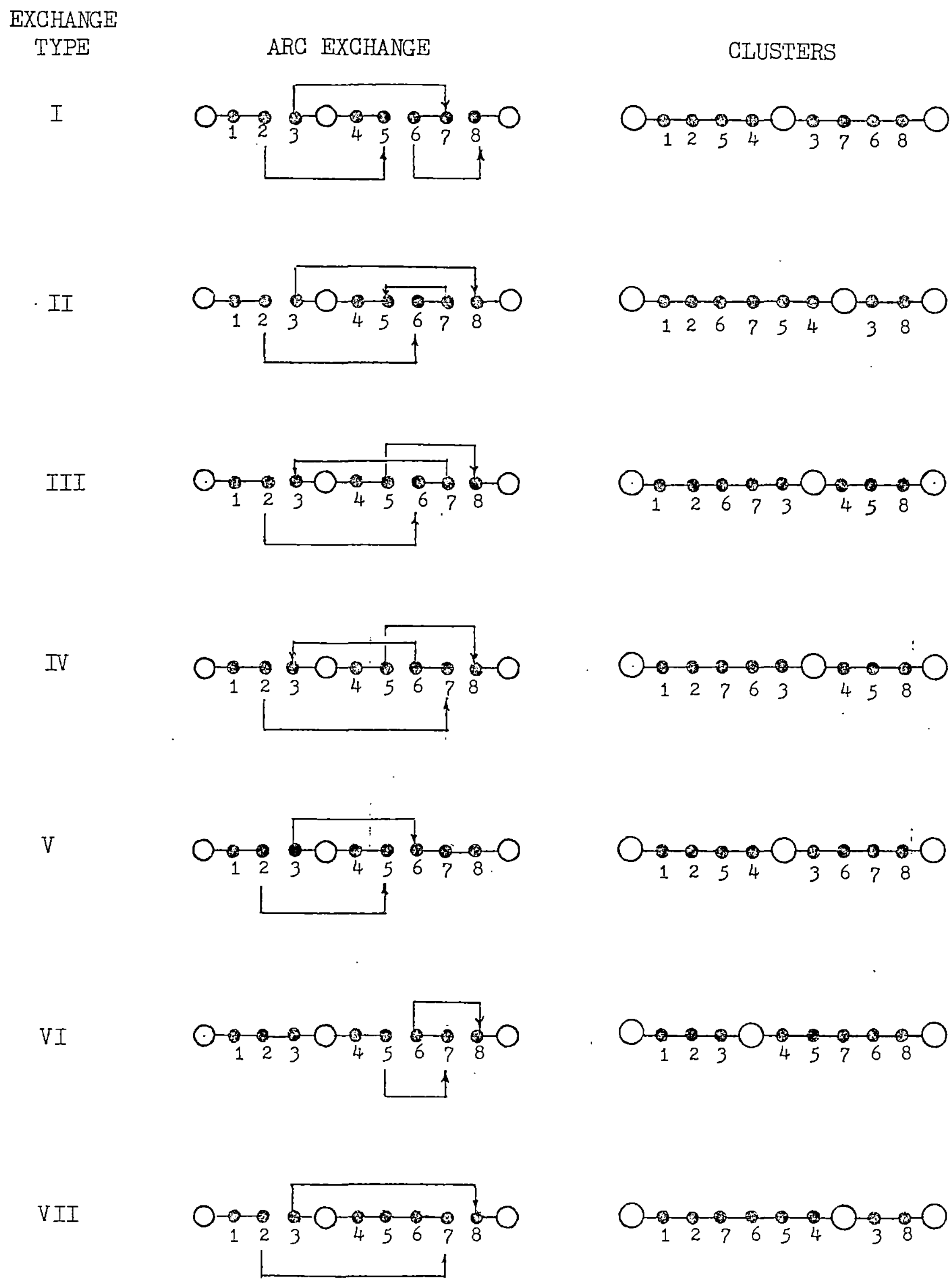


Figure 4.4. Use of Arc Exchanges for Clustering Two Routes

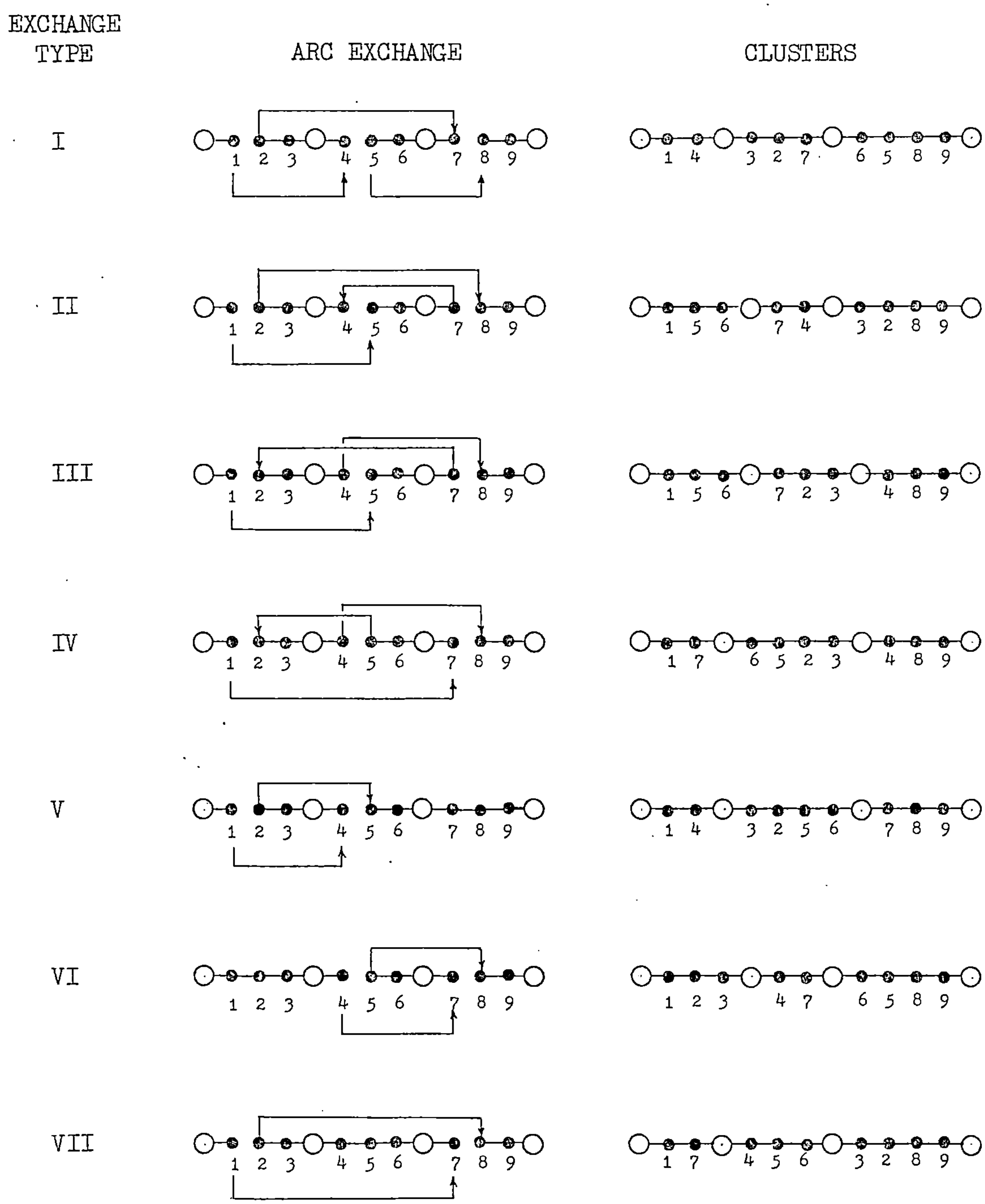


Figure 4.5. Use of Arc Exchanges for Clustering Three Routes

Groups of customers from the middle of each route are not traded in these exchanges. Although this might seem to be a serious limitation on the number of clusters formed, two points should be considered. First, in many cases the most attractive tradeoffs between distance and load involve those customers nearest the depot, since those customers are relatively close to one another. Second, although the immediate result of an exchange does not include trades between the middle of the routes, further exchanges which take place as the algorithm proceeds can produce the same eventual results as if those trades had occurred.

A simplified flow chart for the route-length deviation algorithm is given in Figure 4.6. This flow chart is valid for either 2-arc or 3-arc exchanges. Just as in the total-distance minimization subproblem, the route-length deviation subproblem is solved using 2-arc exchanges prior to being solved using 3-arc exchanges. Whereas the check for improvement in the objective function can be made very early in the total-distance algorithm (see Figure 4.3), this is not possible for the route-length deviation algorithm. TSPs must be solved before an improvement in length deviation can be determined. Because of the computational burden of these TSPs, their solutions are delayed as long as possible in the algorithm. Instead, feasibility checks which can be made without clustering and/or solving TSPs are made in the first part of the algorithm. Any infeasibilities found in these early stages will render the clustering and solving of TSPs unnecessary for the arcs and type of exchange being considered.

Notice that feasibility checks for route loads and route-load deviation can be made without clustering. This is done by keeping track of the cumulative demand, C_i, up to and including customer i, for the

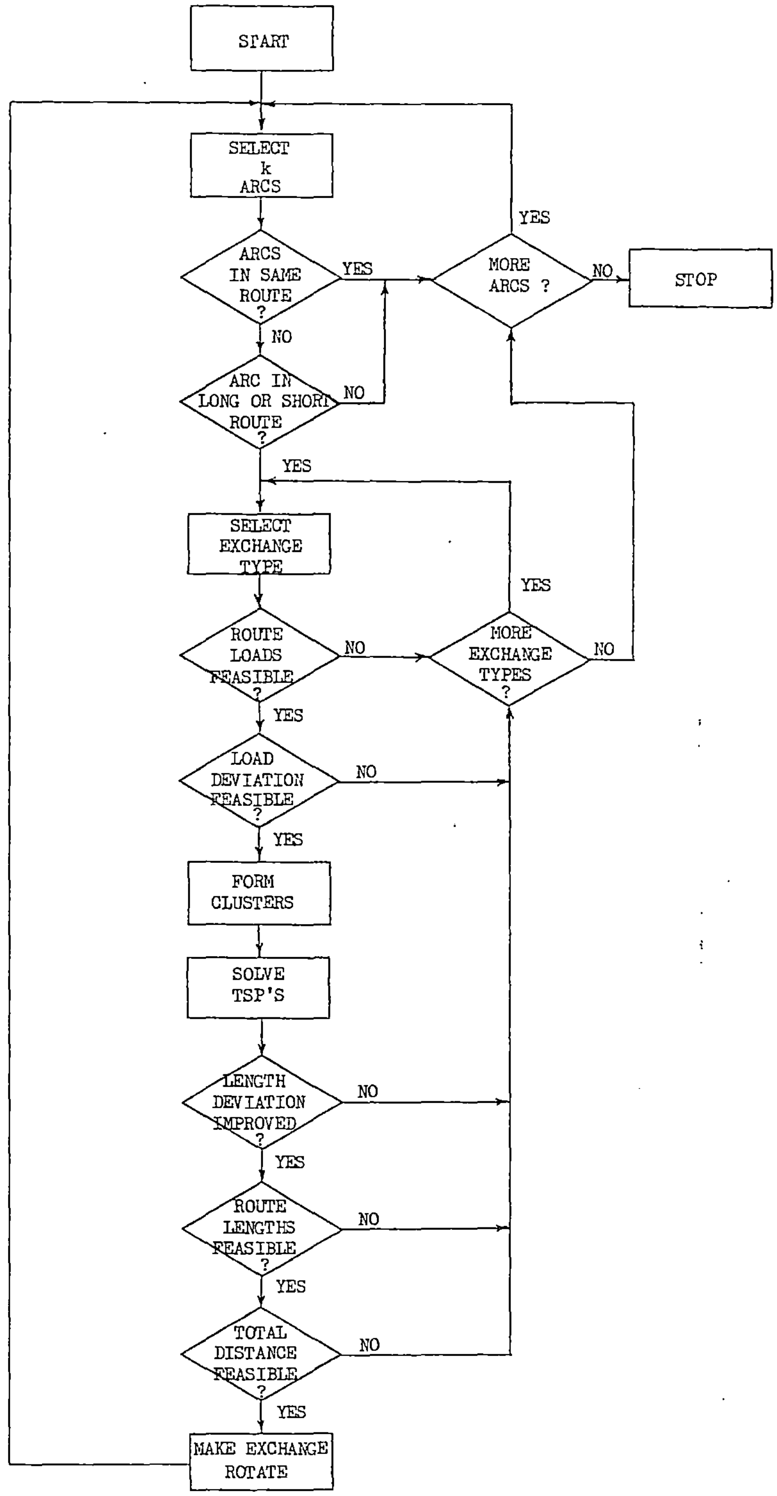


Figure 4.6. Route-Length Deviation Algorithm

route of which i is a member. Consider, for instance, the Type I arc exchange in Figure 4.5. If the current route loads are L_k (k=1, 2, 3), then the three new route loads, L'_k, which will result from the Type I exchange involving arcs (1-2), (4-5), and (7-8) are

$$L'_1 = C_1 + C_4 \tag{4.3}$$

$$L'_2 = L_1 - C_1 + C_7 \tag{4.4}$$

and $$L'_3 = L_2 + L_3 - C_4 - C_7 \tag{4.5}$$

All other route loads in the problem will remain unchanged. Thus, the feasibility checks for route load and route-load deviation can be made without actually making the exchange. Similar rules apply for the other six types of arc exchanges.

Effectiveness of Algorithm

In order to evaluate the quality of solutions obtained from the route-length deviation algorithm, several problems were selected from the literature and solved under different constraining values of total distance, TDIST, and load deviation, LDDEV. The solution chosen as 'best known' in each case was the best of ten runs, obtained by beginning at ten different initial solutions and solving for length deviation, LNDEV. The average percent error measured from this best known solution was 48.83 percent, 28.83 percent, and 18.71 percent for the best of one, two, and three runs, respectively. These percentages seem high in comparison with the errors obtained from the total distance algorithm (see Table 4.1). However, the scale on which the errors are measured has something to do with this. Note that the best known value of LNDEV is less than fifty distance units in ten of the twelve cases, so a small error in absolute units can result in a relatively high error percentage. Another measure of solution quality is given in the table. This is the percent of possible improvement in the initial value of LNDEV obtained from one, two, and three runs. These are shown to be 91.22 percent, 93.40 percent, and 94.72 percent, respectively. Therefore, the algorithm is seen to be capable of making substantial improvement in route-length deviation beginning with an initial least satisfactory achievement level of this objective.

Efficiency of Algorithm

Because the algorithm requires the solution of many TSPs, and because of the computational burden of solving each such TSP, it would seem that the computer time to solve a route-length deviation problem would be highly correlated with the total number of TSPs solved, as well as their size or complexity. To verify this, the number of TSP solutions required in the course of solving each of forty-one different route-length deviation problems of various size was recorded along with the CPU time required for each problem. A regression analysis was performed on the data using a model of the form

$$CPU = \beta_0 \cdot T^{\beta_1} \cdot (N/R)^{\beta_2}, \qquad (4.6)$$

where CPU = computer time in seconds,

T = Number of TSPs solved in problem,

N = Problem size,

and R = Number of routes in problem.

The quantity (N/R) estimates the average size of TSP solved. The model

TABLE 4.3

EFFECTIVENESS OF ROUTE-LENGTH DEVIATION ALGORITHM

Problem Number	Source	No. of Cities	Contraints TDIST	LDDEV	Initial LNDEV	Best of n Runs n=1	n=2	n=3	n=10	Percent Error n=1	n=2	n=3	Percent Improved n=1	n=2	n=3
1	Gaskell (1967)	22	991	4647	95	46	46	45	32	43.75	43.75	40.63	77.78	77.78	79.37
2	Gaskell (1967)	22	1001	4351	95	28	28	28	28	0.00	0.00	0.00	100.00	100.00	100.00
3	Gaskell (1967)	22	1004	2945	162	161	161	161	161	0.00	0.00	0.00	100.00	100.00	100.00
4	Gaskell (1967)	32	849	1727	51	21	21	20	12	75.00	75.00	66.67	76.92	76.92	79.49
5	Gaskell (1967)	32	835	1805	51	33	33	33	33	0.00	0.00	0.00	100.00	100.00	100.00
6	Gaskell (1967)	32	951	283	13	13	13	13	13	0.00	0.00	0.00	100.00	100.00	100.00
7	Christofides and Eilon (1969)	50	577	9	10	4	4	4	4	0.00	0.00	0.00	100.00	100.00	100.00
8	Christofides and Eilon (1969)	50	582	11	53	12	10	7	5	140.00	100.00	40.00	85.42	89.58	95.83
9	Christofides and Eilon (1969)	50	725	3	107	17	17	17	11	54.55	54.55	54.55	93.75	93.75	93.75
10	Christofides and Eilon (1969)	75	870	18	97	57	57	57	57	0.00	0.00	0.00	100.00	100.00	100.00
11	Christofides and Eilon (1969)	75	885	18	97	54	54	54	49	10.20	10.20	10.20	89.58	89.58	89.58
12	Christofides and Eilon (1969)	75	1131	5	81	29	13	9	8	262.50	62.50	12.50	71.23	93.15	98.63
	Average									48.83	28.83	18.71	91.22	93.40	94.72

was linearized to the form

$$\ln (CPU) = \ln (\beta_0) + \beta_1 \ln (T) + \beta_2 \ln (N/R). \qquad (4.7)$$

The results of the regression are given in Table 4.4. Here a strong relationship between the solution time and number of TSPs is seen, with the parameters all being significant at the 0.01 level, and an R^2 of 0.97 being obtained. The model (4.6) becomes

$$CPU = 9.2077 \times 10^{-5} \cdot T^{0.9034} \cdot (N/R)^{1.7279}. \qquad (4.8)$$

Having shown the relationship between the solution time of a route-length deviation problem and the number of TSPs solved in the problem, it is possible to formulate a model for the solution time of a problem if the number of such TSPs can be estimated. This number is a function of three factors:

1. The number of 'qualified' 3-arc combinations encountered in the course of proving 3-arc optimality; i.e., the number of 3-arc combinations having all arcs not in the same route and having arc(s) in the longest route, the shortest route, or both.
2. The degree to which the workload in the current solution is out of balance.
3. The amount of relaxation in the satisfactory achievement levels of the other two objective functions, total distance and route-load deviation.

The number of 'qualified' 3-arc combinations, Q, can be approximated (see Appendix A) to be

$$Q = \frac{(N+R)^3}{6} - \frac{2(N^3-NR^2)}{6R^3} - \frac{(NR+R^2-2N)^3}{6R^3} \qquad (4.9)$$

TABLE 4.4

REGRESSION TABLE FOR SOLUTION TIME OF ROUTE-LENGTH DEVIATION ALGORITHM AS A FUNCTION OF NUMBER OF TSP'S SOLVED

Source	Degrees of Freedom	Sum of Squares	Mean Square	F Value	P(>F)	R^2	Parameter	Estimate	t Value	P(>|t|)
Model	2	17.1912	8.5956	586.49	0.0001	0.97	$\ln(\beta_0)$	-9.2929	-25.51	0.0001
Error	38	0.5569	0.0147				β_1	0.9034	33.92	0.0001
Total (Corrected)	40	17.7481					β_2	1.7279	18.48	0.0001

The degree to which the current solution is out of balance is given by two quantities:

$$B_{Ld} = \frac{LDDEV}{LD_{Max}} \tag{4.10}$$

and

$$B_{Ln} = \frac{LNDEV}{LN_{Max}} \tag{4.11}$$

where B_{Ld} = Route-load imbalance,

B_{Ln} = Route-length imbalance,

LDDEV = Route-load deviation,

LNDEV = Route-length deviation,

LD_{Max} = Maximum load in route set,

and LN_{Max} = Maximum length in route set.

The degree to which the achievement levels of the other two objective functions are relaxed is given by

$$RLX_{dist} = \frac{DLIMIT - DIST}{DIST} \tag{4.12}$$

and $$RLX_{Ld} = \frac{LDDVLM - LDDEV}{C-LDDEV} \tag{4.13}$$

where RLX_{dist} = Total-distance relaxation,

RLX_{Ld} = Load-deviation relaxation,

DLIMIT = Limit on total distance,

DIST = Total distance of current solution,

LDDVLM = Limit on route-load deviation,

and C = Vehicle capacity.

A model for estimating CPU time can be formulated as

$$CPU = \beta_0 \; Q^{\beta_1} \cdot B_{Ln}^{\beta_2} \cdot B_{Ld}^{\beta_3} \cdot RLX_{dist}^{\beta_4} \cdot RLX_{Ld}^{\beta_5} \cdot (N/R)^{\beta_6} \tag{4.14}$$

This model can be linearized through the use of natural logarithms. A preliminary regression analysis of the linear model showed β_2 to be insignificant. The results of the linear model without β_2 are given in Table 4.5. The parameters are all shown to be significant at the 0.10 level, an R^2 of 0.57 being obtained from the model. The final model becomes

$$CPU = 2.3485 \times 10^{-4} \cdot Q^{0.9325} \cdot B_{Ld}^{1.3018} \cdot RLX_{dist}^{0.0686} \cdot RLX_{Ld}^{0.0795} \cdot (N/R)^{2.1574} \qquad (4.15)$$

CPU times on the IBM 3081D ranged from 0.94 seconds for a problem requiring 315 TSPs to 16.10 seconds for a problem requiring 13,101 TSPs. The average solution time for the set of forty-one problems in the analysis was 5.10 seconds, and the standard deviation was 3.17 seconds.

Minimization of Route-Load Deviation

Route-Load Deviation Algorithm

If the decision maker selects route-load deviation as the least satisfactory achievement level in the WBVRP, then route-load deviation must be minimized, subject to the original problem constraints and satisfactory achievement levels of the other two objective functions. The algorithm to do this is based on arc exchanges, as in the algorithm for route-length deviation. The algorithms differ primarily in the arc set on which exchanges can be made and in the order in which feasibility checks are made. The arc exchanges, which are used to cluster customers into new routes, are the same as shown in Figures 4.4 and 4.5 for the route-length deviation algorithm.

A simplified flow chart of the route-load deviation algorithm is

TABLE 4.5

REGRESSION TABLE FOR SOLUTION TIME OF ROUTE-LENGTH DEVIATION ALGORITHM AS A FUNCTION OF PROBLEM SIZE, NUMBER OF ROUTES, WORKLOAD IMBALANCE, AND ACHIEVEMENT LEVEL RELAXATION

Source	Degrees of Freedom	Sum of Squares	Mean Square	F Value	P(>F)	R^2	Parameter	Estimate	t Value	P(>|t|)
Model	5	10.1140	2.0228	9.27	0.0001	0.57	$\ln(\beta_0)$	-8.3566	-3.65	0.0009
Error	35	7.6341	0.2181				β_1	0.9235	4.71	0.0001
Total (Corrected)	40	17.7481					β_3	1.3018	4.75	0.0001
							β_4	0.0686	1.94	0.0608
							β_5	0.0795	3.24	0.0027
							β_6	2.1574	3.28	0.0024

shown in Figure 4.7. The similarity with the route-length deviation algorithm can be seen by comparing this flow chart with the one in Figure 4.6. Whereas the clustering step is preceded by a check for load-deviation feasibility in the route-length deviation algorithm, the clustering step is preceded by a check for load-deviation improvement in the route-load deviation algorithm. Under similar conditions, therefore, the route-load deviation algorithm can be expected to require fewer TSP solutions than the route-length deviation algorithm, since an arc exchange is less likely to result in an absolute improvement in a current value of route-load deviation than in a value of route-load deviation which is less than or equal to a relaxed limit on this achievement level. In effect, the feasibility check in Figure 4.7 is a better 'TSP filter' than the improvement check in Figure 4.6.

Effectiveness of Algorithm

To evaluate the quality of solutions produced by the route-load deviation algorithm, several problems were selected from the literature and solved for route-load deviation, LDDEV, under different constraining levels of total distance, TDIST, and route-length deviation, LNDEV. Each problem was solved ten times, and the best of one, two, and three runs was recorded and compared with the best of ten runs (the assumed 'best known' solution). Table 4.6 contains the results of this analysis. The error was found to be 43.07 percent, 37.30 percent, and 24.36 percent for the best of one, two, and three runs, respectively. Just as in the route-length deviation problems shown in Table 4.3, the high error percentages are due in part to the scale involved. For instance, the Christofides and Eilon problems in the table have load-deviations

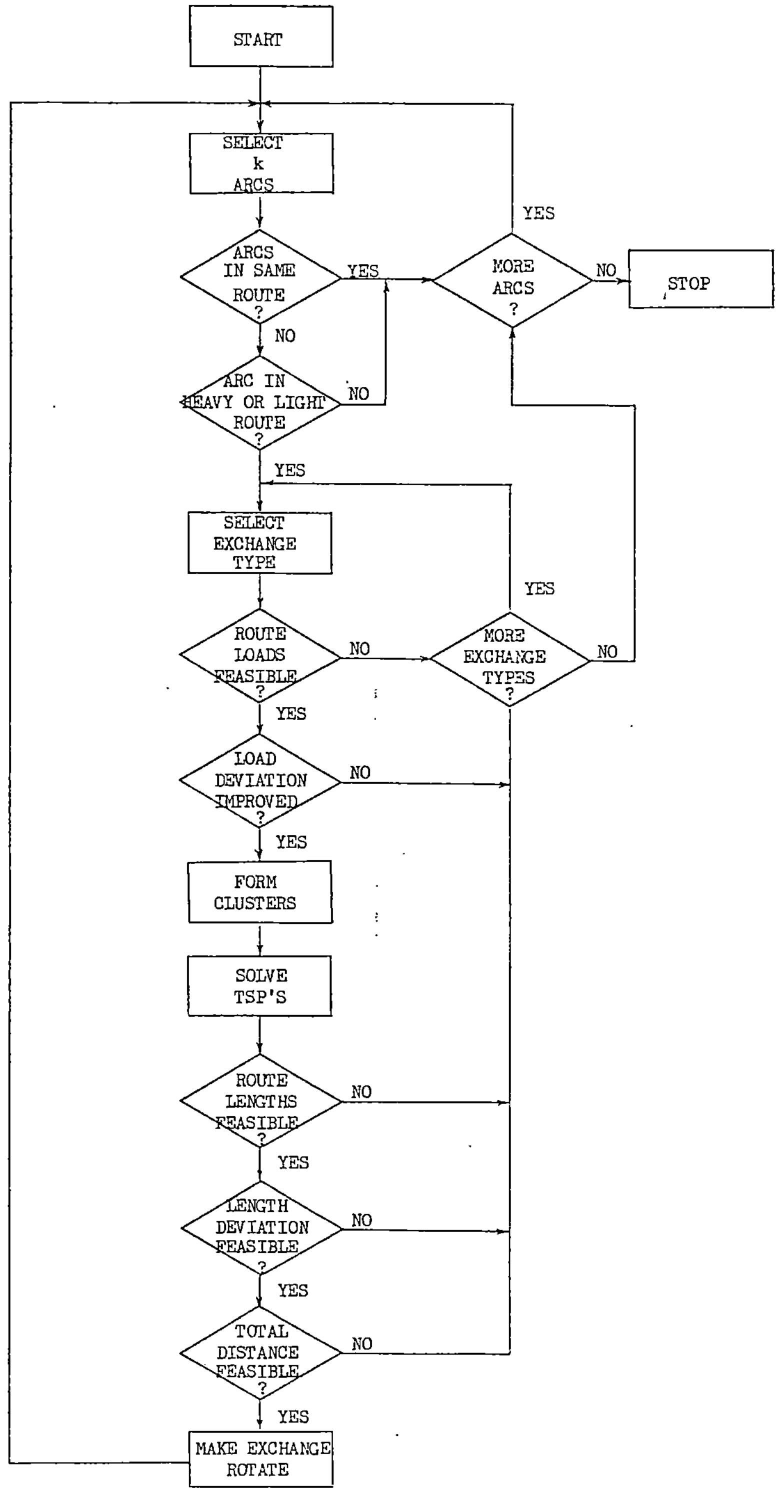


Figure 4.7. Route-Load Deviation Algorithm

TABLE 4.6

EFFECTIVENESS OF ROUTE-LOAD DEVIATION ALGORITHM

Problem Number	Source	No. of Cities	Constraints TDIST	Constraints LNDEV	Initial LDDEV	Best of n Runs n=1	n=2	n=3	n=10	Percent Error n=1	n=2	n=3	Percent Improved n=1	n=2	n=3
1	Gaskell (1967)	22	1095	8	3511	3511	3511	3511	3511	0.00	0.00	0.00	100.00	100.00	100.00
2	Gaskell (1967)	22	977	99	4225	3275	3275	3275	3275	0.00	0.00	0.00	100.00	100.00	100.00
3	Gaskell (1967)	22	986	97	4225	3256	3256	3105	3105	4.86	4.86	0.00	86.52	86.52	100.00
4	Gaskell (1967)	32	872	13	1770	750	750	750	750	0.00	0.00	0.00	100.00	100.00	100.00
5	Gaskell (1967)	32	827	53	1570	700	700	700	700	0.00	0.00	0.00	100.00	100.00	100.00
6	Christofides and Eilon (1969)	50	702	3	9	3	3	3	3	0.00	0.00	0.00	100.00	100.00	100.00
7	Christofides and Eilon (1969)	50	586	60	8	1	1	1	1	0.00	0.00	0.00	100.00	100.00	100.00
8	Christofides and Eilon (1969)	50	596	11	7	5	5	5	3	66.67	66.67	66.67	50.00	50.00	50.00
9	Christofides and Eilon (1969)	50	582	6	9	4	4	3	3	33.33	33.33	0.00	83.33	83.33	100.00
10	Christofides and Eilon (1969)	75	919	109	17	7	6	6	4	75.00	50.00	50.00	76.92	84.62	84.62
11	Christofides and Eilon (1969)	75	914	108	17	8	7	6	2	300.00	250.00	200.00	60.00	66.67	73.33
12	Christofides and Eilon (1969)	75	1013	12	10	7	7	5	5	40.00	40.00	0.00	60.00	60.00	100.00
13	Christofides and Eilon (1969)	75	1013	14	10	7	7	5	5	40.00	40.00	0.00	60.00	60.00	100.00
	Average									43.07	37.30	24.36	82.83	83.93	92.92

measured in hundredweight units, and the very least error which could occur in these problems is twenty percent, given that there is an error at all. Because of this scaling problem, another measure of solution quality is included in the table. This is the percent improvement made in the initial solution, measured against the possible improvement in going from the initial LDDEV to the best of ten runs. This percent improvement is seen to be 82.83 percent, 83.93 percent, and 92.92 percent for the best of one, two, and three runs, respectively.

Efficiency of Algorithm

As stated above, the check for improvement in route-load deviation in Figure 4.7 provides a better 'TSP filter' than does the check for feasibility of route-load deviation in Figure 4.6. Thus, the solving of fewer TSPs should be required by the route-load deviation algorithm than by the route-length deviation algorithm. This, in turn, should cause the relationship between the total solution time and the number of TSPs solved to be weaker than in the route-length deviation algorithm. To verify this, a regression analysis was performed on data from forty-seven route-load deviation problems using a model of the form

$$CPU = \beta_0 \cdot T^{\beta_1} \cdot (N/R)^{\beta_2} \qquad (4.16)$$

where CPU = computer time in seconds,

T = Number of TSPs solved in problem,

N = Problem size,

and R = Number of routes in problem.

As before, (N/R) is an estimate of the average size of TSP solved. A preliminary analysis on a linear (logarithmic) version of the model showed β_2 to be insignificant. The results of a regression analysis on a logarithmic model without β_2 is given in Table 4.7. Using these results, equation (4.16) is rewritten as

$$CPU = 0.3239\ T^{0.2028} \tag{4.17}$$

As suspected, the R^2 value of 0.33 is smaller than the R^2 value for the route-length deviation model, which was found to be 0.97 (see Table 4.4). Nonetheless, since the solution time depends to some extent upon the number of TSPs solved, a model similar to equation (4.14) can be written as

$$CPU = \beta_0 \cdot Q^{\beta_1} \cdot B_{Ld}^{\beta_2} \cdot B_{Ln}^{\beta_3} \cdot RLX_{dist}^{\beta_4} \cdot RLX_{Ln}^{\beta_5} \cdot (N/R)^{\beta_6} \tag{4.18}$$

where $RLX_{Ln} = \dfrac{LNDVLM - LNDEV}{LNDEV}$ (4.19)

and the other terms are the same as defined previously. A regression analysis on a logarithmic version of (4.18) showed only β_0, β_1, β_2, and β_6 to be significant. The results of a final analysis on the logarithmic model are given in Table 4.8. From this table, equation (4.18) can be rewritten as

$$CPU = 6.8664 \times 10^{-4} \cdot Q^{0.7566} \cdot B_{Ld}^{1.0111} \cdot (N/R)^{1.1964} \tag{4.20}$$

The parameters β_0, β_1, β_2, and β_6 are significant at the 0.05 level, the model obtaining an R^2 of 0.47. Comparing this with the results of the route-length deviation model (Table 4.5), which has an R^2 of 0.57, it is seen that the route-load deviation model is not as capable of explaining the variation in solution times. The CPU times for route-load deviation problems ranged from a low of 0.16 seconds to a high of 5.39 seconds, with an average of 1.30 seconds and a standard deviation of 1.12 seconds.

TABLE 4.7

REGRESSION TABLE FOR SOLUTION TIME OF ROUTE-LOAD DEVIATION ALGORITHM AS A FUNCTION OF NUMBER OF TSP'S SOLVED

Source	Degrees of Freedom	Sum of Squares	Mean Square	F Value	P(>F)	R^2	Parameter	Estimate	t Value	P(>|t|)
Model	1	9.5408	9.5408	22.61	0.0001	0.33	$\ln(\beta_0)$	-1.1273	-4.62	0.0001
Error	45	18.9879	0.4220				β_1	0.2028	4.76	0.0001
Total (corrected)	46									

TABLE 4.8

REGRESSION TABLE FOR SOLUTION TIME OF ROUTE-LOAD DEVIATION ALGORITHM AS A FUNCTION OF PROBLEM SIZE, NUMBER OF ROUTES, AND WORKLOAD IMBALANCE

Source	Degrees of Freedom	Sum of Squares	Mean Square	F Value	P(>F)	R^2	Parameter	Estimate	t Value	P(>|t|)
Model	3	13.3865	4.4622	12.67	0.0001	0.47	$\ln(\beta_0)$	-7.2837	-4.26	0.0001
Error	43	15.1422	0.3521				β_1	0.7566	4.55	0.0001
Total (corrected)	46	28.5287					β_2	1.0111	5.80	0.0001
							β_6	1.1964	2.15	0.0375

Summary

In this chapter three different single-objective algorithms, necessary to the implementation of the Method of Satisfactory Goals in solving the WBVRP, have been presented. These algorithms are used to find minimum values of total distance, route-length deviation, and route-load deviation, each being subject to the original problem constraints and satisfactory achievement levels of the other two objectives. Computational experience was used to provide an evaluation of the effectiveness and efficiency of each algorithm. Tradeoffs between the effectiveness and efficiency of each are necessary in implementing the algorithm in an interactive computer program to solve the WBVRP. This interactive computer program is the subject of the next chapter.

CHAPTER V

INTERACTIVE COMPUTER PROGRAM

Introduction

This chapter contains a description and evaluation of an interactive computer program written to solve the workload-balanced vehicle routing problem (WBVRP). The single-objective algorithms presented in Chapter IV are used in the program to implement a heuristic version of the Method of Satisfactory Goals (Benson, 1975). The program was written in IBM VS FORTRAN and compiled under level two optimization on the IBM 3081D at Oklahoma State Universtiy. Graphics display capability is provided by the Tektronix Plot 10 Terminal Control System. All displays illustrated in this chapter are from a Tektronix 4105 graphics display terminal.

Reeves and Franz (1985) list six criteria deserving specific attention in the development of interactive approaches. These are summarized as follows:

1. Minimize required inputs, such as weights or other quantitative assessments, from the decision maker.
2. Simplify the process by, for example, reducing the alternatives presented to the decision maker at each iteration as much as possible.
3. Provide for backtracking, realizing that learning behavior may occur during interaction with the model.
4. Allow the decision maker to reach a satisficing solution in

relatively few steps, realizing that in an interactive process it is not meaningful to expect an exact optimal solution.

5. Structure the choice of alternatives so they are similar at each step, allowing the decision maker to continue using a familiar decision process.

6. Enable the solution of large-scale, real world problems by avoiding methodologies that require the generation of the complete efficient solution set or are otherwise unnecessarily complex computationally.

It is felt that the interactive program described in this chapter meets all of the criteria listed above.

Program Description

Menu

The primary interaction between the decision maker and the program occurs through an on-screen menu which allows the decision maker to specify one of eight different functions. The menu is one of three different displays which appear together on the screen. The other two displays are (1) problem status and (2) tradeoff information, which are illustrated later as a sample problem is solved.

A typical screen display is shown in Figure 5.1. The menu, which appears on the left of the screen, offers eight options. They are:

1. Minimize Total Distance. This option causes a distance minimization problem to be solved, subject to the original problem constraints and satisfactory achievement levels of route-load deviation and route-length deviation (for which the decision maker will subsequently be prompted). Selection of

this option can be the result of a change in the objective function having the least satisfactory achievement level, or can be the result of an insufficient improvement in total distance during the previous iteration.

2. Minimize Load Deviation. A route-load deviation problem is to be solved, subject to the original problem constraints and satisfactory achievement levels of total distance and route-length deviation.

3. Minimize Length Deviation. A route-length deviation problem is to be solved, subject to the original problem constraints and satisfactory achievement levels of total distance and route-load deviation.

4. Manual Route Improvement. The decision maker is to make an adjustment to a route structure by specifying a new ordering of customers in the route. The program evaluates the effects of the change on the route length. If an improvement is made by the adjustment, the changes are implemented; otherwise the manual adjustment is ignored.

5. Display Previous Solution. A prior route structure is displayed graphically, along with the values of each objective function's achievement level. No backtracking occurs if this item is selected.

6. Backtrack to Previous Solution. As in menu item (5), a prior route structure and all of the achievement levels for that solution are displayed. In addition, all of the problem characteristics are reset to the values as contained in that prior solution.

7. Remove Route from Calculations. This option enables one or more routes to be ignored in calculating route-load and route-length deviation. This can be used, for instance, in the case of a very long route serving isolated customers, the inclusion of which would cause unacceptable tradeoffs in attempting to minimize route-length deviation.
8. Exit. The program is terminated, usually after an acceptable solution has been found.

<u>Multiple Runs of Single-Objective Algorithms</u>

Selection of menu items (1), (2), or (3) calls for solution of a single-objective problem. Any arc-exchange algorithm, whether it be a k-opt distance minimization algorithm or a k-arc deviation minimization algorithm, should be run more than once in order to improve the chances that a good solution is reached. To determine the number of runs, a tradeoff between the solution quality and the expense of multiple runs must be reached. The result of the analyses of Chapter IV are used in making this tradeoff. The average CPU time to solve a distance minimization problem (excluding the 100-city problems, for which no solution times were obtained for the other two objectives) was 1.86 seconds, the average time for a route-load deviation problem was 1.30 seconds, and the average time for a route-length deviation problem was 5.10 seconds. The average number of these subproblems required in the solution of a WBVRP was found over a sample of WBVRPs to be almost six. In order to limit the total CPU time to a reasonable amount, the following number of runs for each single-objective algorithm was established for the program:

```
                    GASKELL'S 22-CITY PROBLEM
                      SOLUTION NUMBER  3

        MAIN MENU                  STATUS  LIMIT          ROUTES:
                                                        #  LOAD DIST
1. MINIMIZE TOTAL DISTANCE     --    994     994        1 1144   227
2. MINIMIZE LOAD DEVIATION     --   3100    ****        2    0     0
3. MINIMIZE LENGTH DEVIATION   --     88      95        3 2400   183
4. MANUAL ROUTE IMPROVEMENT                             4 4225   140
5. DISPLAY PREVIOUS SOLUTION                            5 1295   216
6. BACKTRACK TO PREV. SOL.                              6 1125   228
7. REMOVE ROUTE FROM CALC.
8. EXIT
                     ESTIMATED TRADEOFFS:
                                  1    2    3    4    5
  LOAD DEVIATION   IMPROVEMENT   144  125   41   19    5
  TOTAL DISTANCE   RELAXATION    -39  -25   31   21   18
  LENGTH DEVIATION RELAXATION     73   62    0    3    9
  ORIGINAL TRADEOFFS  34    REDUCED TRADEOFFS   5
SELECT FROM MENU
```

Figure 5.1. Screen Display Containing Menu, Problem Status, and Tradeoff Information

1. Total Distance: three runs

2. Route-Load Deviation: three runs

3. Route-Length Deviation: two runs

From the results of the single-objective effectiveness analysis (Tables 4.1, 4.3, and 4.6), the following solution quality can be expected from the multiple runs of the algorithms:

1. Total Distance: 3.14% error

2. Route-Load Deviation: 92.92% improvement

3. Route-Length Deviation: 93.40% improvement

Using these criteria, an 'average' six-iteration problem could be expected to require a total of about forty seconds CPU time, not including the time required to reach a beginning solution.

The mechanism for finding alternate starting points for the multiple runs is straightforward. After the first run, the achievement level of the objective function is increased by a factor. The algorithm is then begun, trying to improve this relaxed achievement level. As soon as a feasible arc-exchange is discovered which improves the relaxed achievement level, the route structure is altered and this newly found value of the objective function becomes the value to be improved. In subsequent iterations of the single-objective algorithm, no relaxation of the achievement level is applied. This is seen to be a primal approach, since the problem always remains feasible.

Goal Tradeoffs

In the Method of Satisfactory Goals, the decision maker must select the amount by which one or more goals can be relaxed in order to improve the least satisfactory achievement level (if the last iteration has not yielded sufficient improvement). Values of dual variables are provided for this purpose. In the heuristic version of the method used to solve the WBVRP, no such values of dual variables can be provided. Some other means of estimating the effects of goal relaxation must be used, instead. The method employed in the interactive computer program involves the calculation of goal tradeoffs during the final 'proving' stage of the single-objective algorithm.

Suppose, in evaluating a particular arc exchange, the exchange is found to be feasible with respect to the original problem constraints but infeasible with respect to one or both of the two constraining achievement levels. The flow charts in Figures 4.3, 4.6, and 4.7 indicate that the evaluation of that particular arc exchange would be abandoned and another set of arcs selected for evaluation. To provide tradeoff information, however, the evaluation is not abandoned, but continues as if the arc exchange were feasible. As a final step, the program calculates the amount of goal relaxation required to make the exchange feasible. The amount of improvement in the objective function and the amount of relaxation in the constraining achievement levels are stored as a single tradeoff, along with the arcs involved in the exchange. All such tradeoffs are kept in an array for display to the decision maker. The array is reduced by eliminating any dominated exchanges (those exchanges providing lesser improvements in the objective function for greater relaxation in the constraints). The reduced (nondominated) tradeoffs are sorted in order of decending amount of objective function improvement and displayed to the decision maker, up to a maximum of six such tradeoffs. A typical set of such tradeoffs is shown in the lower portion of Figure 5.1.

A tradeoff will require the relaxation of one or both of the two constraining achievement levels. If both relaxations indicated in a given tradeoff have positive values, then both constraints will be increased. If one of the relaxations is negative, the <u>immediate</u> result of the arc exchange is an improvement in that particular achievement level. However, that constraint is not tightened by the program, since the Method of Satisfactory Goals does not operate by tightening constraints. The constraint on the achievement level showing a negative relaxation actually remains unchanged during the solution of the subsequent single-objective problem. The decision maker must realize this when choosing a tradeoff. Only positive relaxations should be used in making this choice; negative relaxations should be considered only when trying to decide between two or more tradeoffs which are otherwise equally attractive.

The tradeoffs provided by the program are only local estimates of the effects of constraint relaxation. After an arc exchange corresponding to the tradeoff is made, the algorithm attempts to make further improvements in the objective function using the new achievement level constraints. If further improvements can be made, then the effect of the tradeoff has been underestimated by the tradeoff. Table 5.1 contains the results of various tradeoffs made in the course of solving 28 different problems. The table shows the percent improvement 'promised' by the tradeoff, the percent improvement actually obtained, and the ratio of the two, expressed as an 'improvement ratio'. Of the 28 tradeoffs in the table, 13 resulted in actual improvements which were the same as promised by the tradeoffs. The remainder resulted in improvements better than indicated by the tradeoffs. The average improvement ratio from the problems in Table 5.1 is 1.35, and the standard deviation is 0.67. Now, it

TABLE 5.1

ACTUAL VERSUS PROMISED IMPROVEMENT IN OBJECTIVE FUNCTION PROVIDED BY TRADEOFFS

Problem Number	No. of Cities	Objective Function TDIST	LNDEV	LDDEV	Percent Constraint Relaxation TDIST	LNDEV	LDDEV	Percent Promised Improvement	Percent Actual Improvement	Improvement Ratio
1	22	X				62.00		2.74	2.74	1.00
2	22	X					5.38	0.21	0.31	1.48
3	22		X		11.28			60.00	90.53	1.51
4	22	X				350.00		3.28	5.21	1.59
5	22	X				55.56	2.50	1.02	1.02	1.00
6	22			X	7.92			10.87	13.62	1.25
7	22			X		245.00		5.87	5.87	1.00
8	32			X	3.33	11.76		50.32	50.32	1.00
9	32			X	1.44			65.38	65.38	1.00
10	32			X	0.24	15.91		14.81	25.93	1.75
11	32	X				650.00		3.65	4.43	1.21
12	32			X	0.55	80.00		51.18	51.18	1.00
13	32			X		175.00		34.00	34.00	1.00
14	32	X				9.62	70.00	0.23	0.23	1.00
15	50		X		3.28		37.50	12.35	18.52	1.50
16	50		X		12.54		54.55	54.55	86.36	1.58
17	50		X		0.34		140.00	35.29	35.29	1.00
18	50		X		2.90			45.45	45.45	1.00
19	50		X		1.00		25.00	16.67	33.33	2.00
20	50			X		75.00		42.86	57.14	1.33
21	50			X		171.43		33.33	33.33	1.00
22	75			X	3.53	4.35		25.00	37.50	1.50
23	75			X	3.68			20.00	20.00	1.00
24	75	X					125.00	3.15	3.72	1.18
25	75	X				3.19	11.11	0.12	0.12	1.00
26	75		X		6.79			43.62	54.26	1.24
27	75		X		3.18		55.56	46.51	53.49	1.15
28	75		X		1.38			10.00	45.00	4.50

would be desirable to have a small standard deviation in the improvement ratio. Then the decision maker could be confident that the problem is progressing in the right direction. A large standard deviation increases the chances that the decision maker will choose a tradeoff which results in a solution less than otherwise desirable; that is, a tradeoff not selected could have resulted in greater improvement.

Flow Chart

Figure 5.2 shows a simplified flow chart of the interactive program's main routine. The following variables are used in this flow chart:

DLIMIT = Limit (constraint) on total distance.

LDDEV = Route-load deviation.

LDDVLM = Limit on route-load deviation.

LNDEV = Route-length deviation.

LNDVLM = Limit on route-length deviation.

OBJ = Objective function solved in the previous iteration:

(1). Total distance

(2). Route-load deviation

(3). Route-length deviation

TDIST = Total distance (sum of all route lengths).

Only those subroutines called directly by the main program are shown in Figure 5.2. These subroutines (excluding utility functions and Plot 10 graphics routines) perform the following functions:

ADJUST - Accepts decision maker's manual route adjustments to the route structure, and evaluates the effects of those adjustments.

BKTRAK - Backtracks to a specified prior solution.

DISPLA - Displays (only) a specified prior solution.

LDDV2 - Minimizes route-load deviation using 2-arc exchanges.

LDDV3 - Minimizes route-load deviation using 3-arc exchanges.

LNDV2 - Minimizes route-length deviation using 2-arc exchanges.

LNDV3 - Minimizes route-length deviation using 3-arc exchanges.

LOCK - Excludes (locks out) one or more routes in calculation of route-load deviation and route-length deviation.

NONDOM - Reduces set of tradeoffs by eliminating dominated tradeoffs.

SAVNGS - Minimizes total distance using Clarke and Wright's savings algorithm.

TWOOPT - Minimizes total distance using a 2-opt arc exchange algorithm.

THROPT - Minimizes total distance using a 3-opt arc exchange algorithm.

The main program depicted in Figure 5.2 represents a computer implementation of the general WBVRP model structure of Figure 3.2. Note that the first solution presented to the decision maker (first page of Figure 5.2) is the best of eight total distance minimization solutions, those solutions being obtained by successive implementations of the Clarke and Wright savings algorithm, the 2-opt algorithm, and the 3-opt algorithm. Alternate starting solutions are obtained by randomly mixing up the order of the first few elements of the savings file (those elements responsible for initial route formation). This initial solution is the only one which is chosen from so many iterations of a single-objective algorithm. The reason for doing so is to try to reach a satisfactory solution to begin with, an assumption being that the decision maker will consider a minimum-distance solution a satisfactory starting point. After this initial solution, the decision maker is given only three choices: (1) accept the solution as a final one, (2) minimize route-load deviation, or (3) minimize route-length deviation. The minimization of total distance is not a choice here, since it is assumed that a minimum distance solution has been obtained by the eight iterations of the distance minimization algorithms. After this, the program will always return to the menu display on the second page of the flow chart, ultimately terminating when the decision maker selects the last menu option.

Evaluation of Interactive Computer Program

To evaluate the performance of the interactive program, two areas are considered. First is the effectiveness of the procedure, or its ability to generate good solutions. Second it is efficiency, or time required to reach a final solution.

Effectiveness of Program

Convergence Analysis. One of the ways of evaluating the effectiveness of a procedure is to determine whether the procedure will converge to the same final solution from different starting points. In order to demonstrate this, some means of insuring consistency on the part of the decision maker must be provided. To do this, a linear additive utility function is assumed. Recall that the Method of Satisfactory Goals does not assume any type of utility function; the only reason for using one here is to provide an objective means of evaluating the

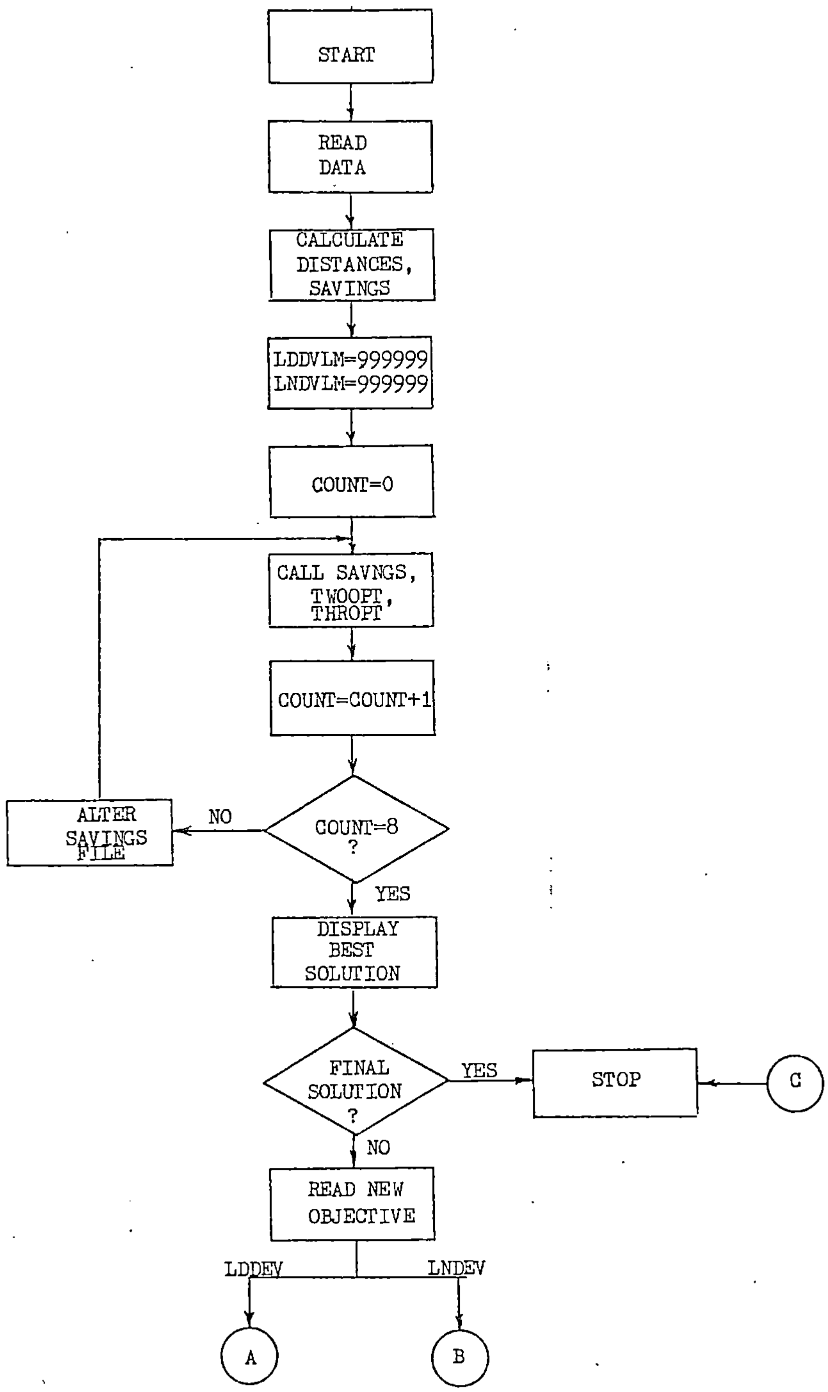


Figure 5.2. Flow Chart for Main Program

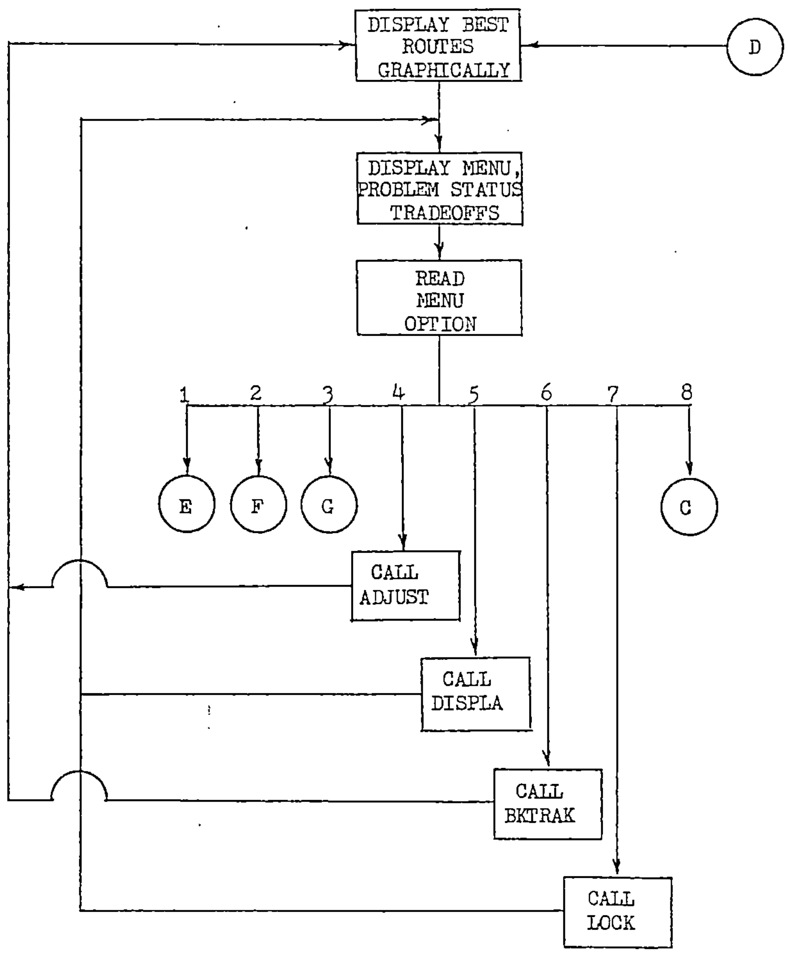


Figure 5.2. (Continued)

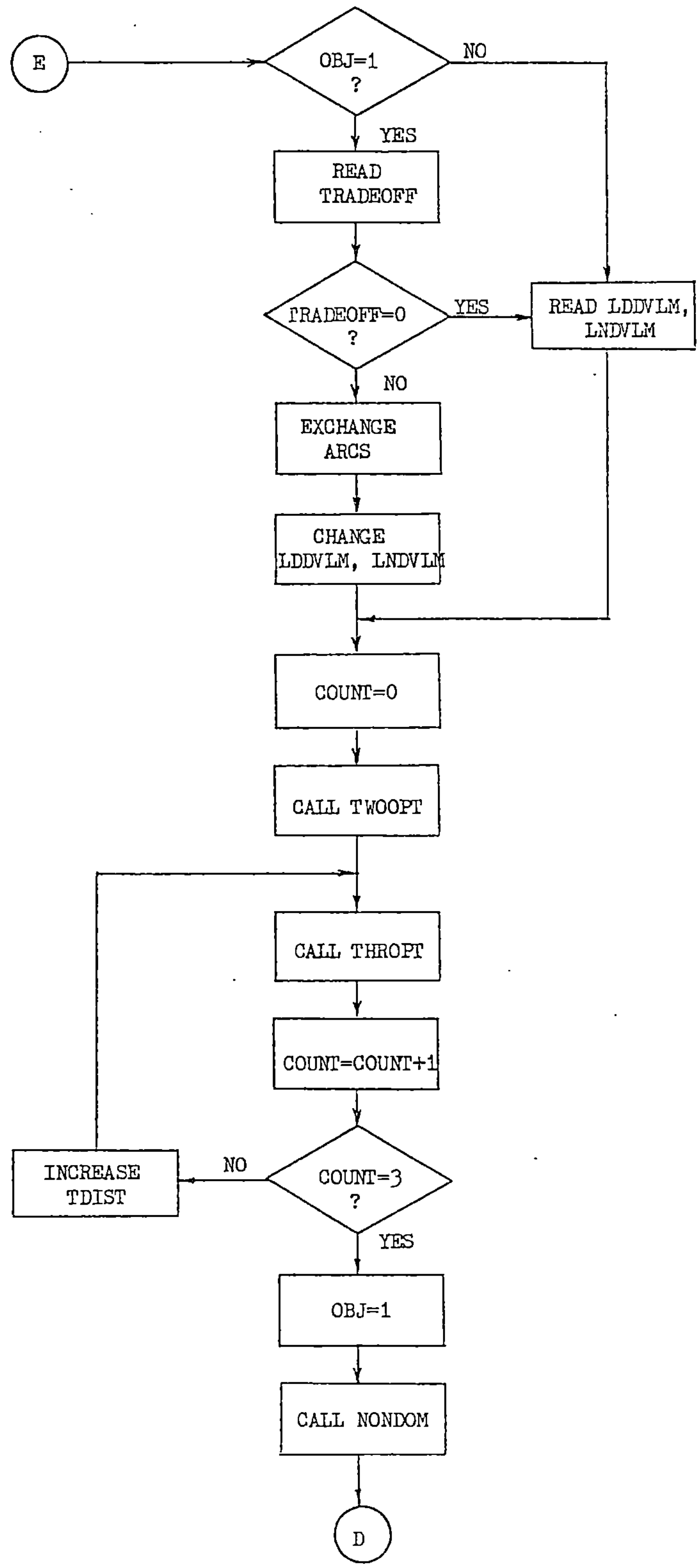


Figure 5.2. (Continued)

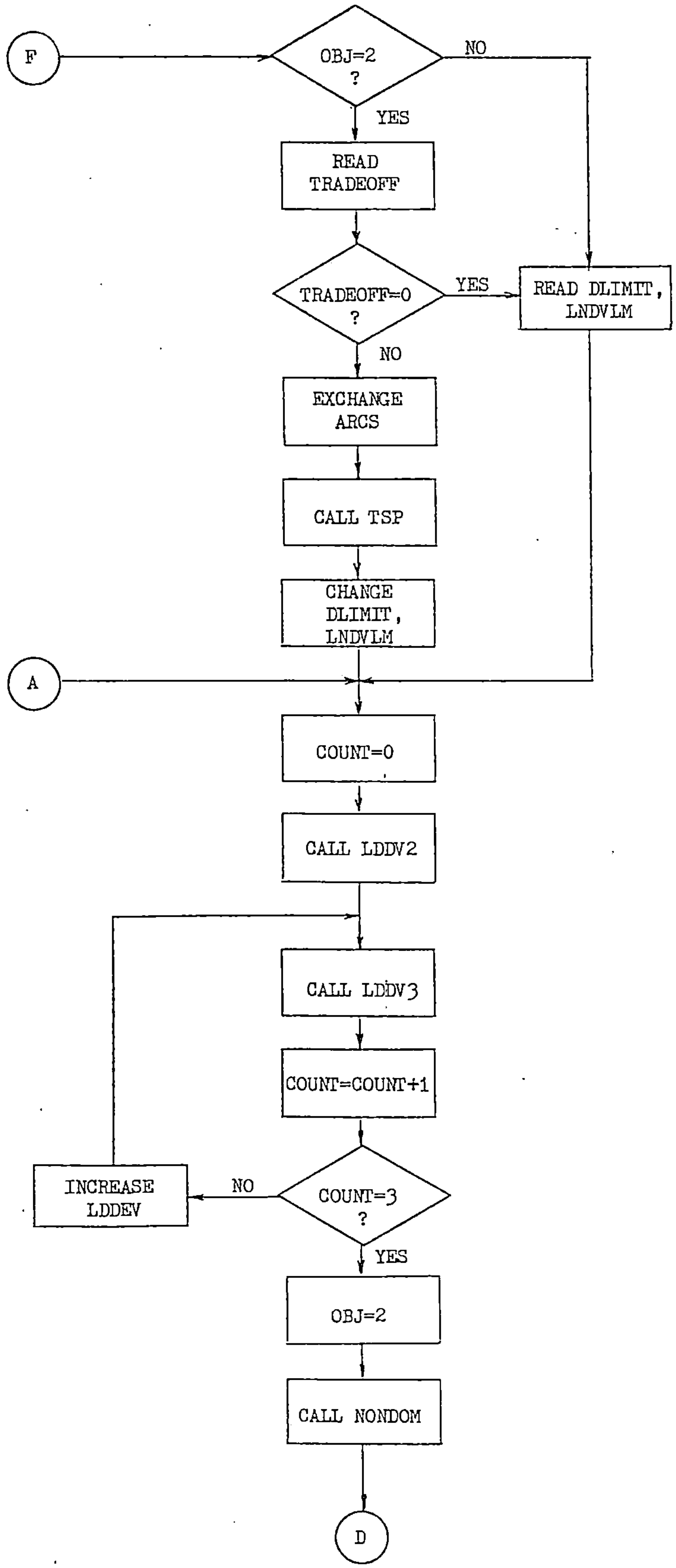


Figure 5.2. (Continued)

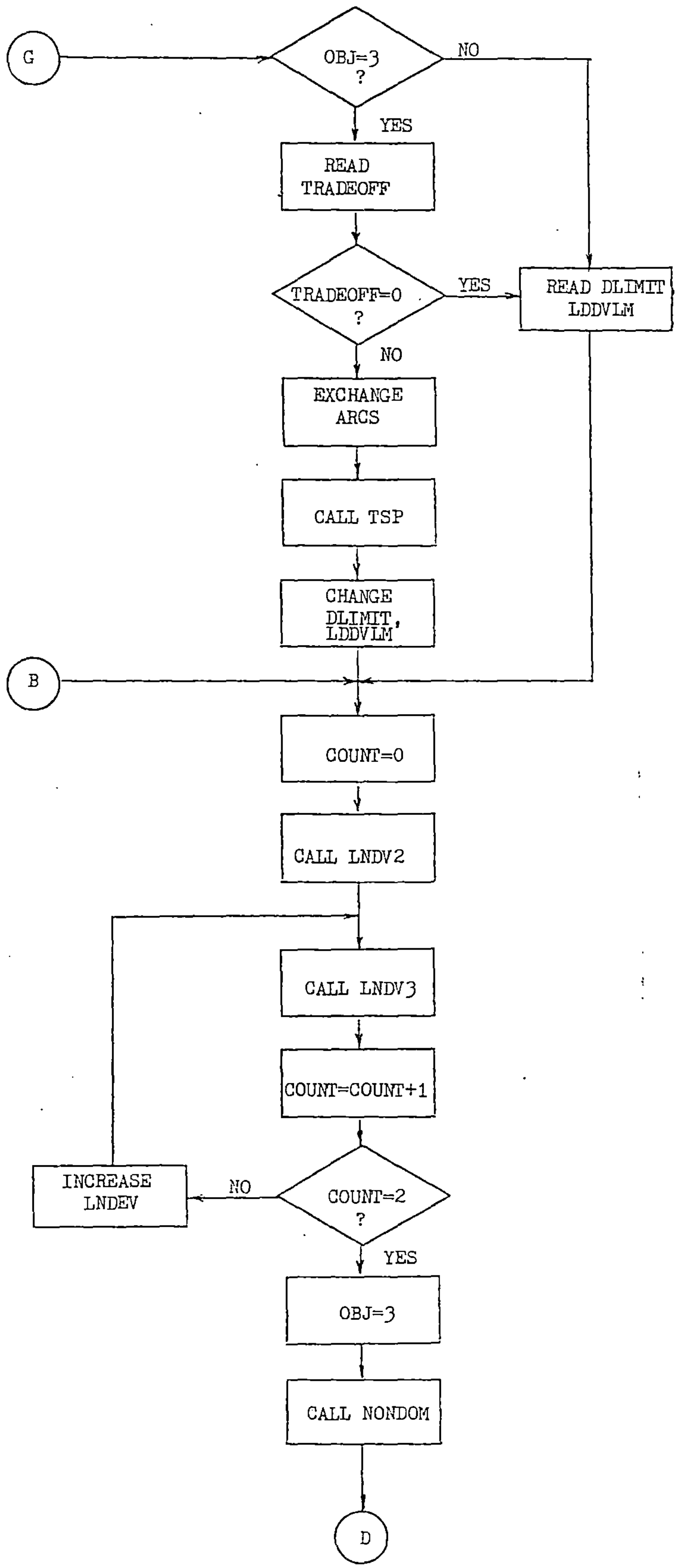


Figure 5.2. (Continued)

interactive procedure, free from inconsistencies introduced by the decision maker's choices.

To formulate the utility function, the range for each objective function is first scaled. For the ith objective, call the best (Ideal) solution value obtained for that objective I_i, and call its worst (Anti-Ideal) solution value A_i. The range of values in going from that objective's worst solution value to its best solution value is

$$R_i = A_i - I_i \tag{5.1}$$

A scaling factor, α_i, is applied to each objective function. Let α_1 be applied to total distance, α_2 to route-load deviation, and α_3 to route-length deviation. The scaling factors are found by simultaneously solving the following three equations:

$$\alpha_1 R_1 - \alpha_2 R_2 = 0, \tag{5.2}$$

$$\alpha_2 R_2 - \alpha_3 R_3 = 0, \tag{5.3}$$

and $$\alpha_1 + \alpha_2 + \alpha_3 = 1. \tag{5.4}$$

For an unequally weighted utility function (e.g., a 3-2-1 weighting on the three objectives), α_1 is replaced by 3 x α_1, α_2 is replaced by 2 x α_2, and the new weights are again normalized to sum to unity. The utility function is written

$$U = \alpha_1 \cdot \text{TDIST} + \alpha_2 \cdot \text{LDDEV} + \alpha_3 \cdot \text{LNDEV}. \tag{5.5}$$

To determine the least satisfactory achievement level for purposes of the analysis, the current achievement level is measured against the Ideal solution for each objective function. Call this the measure of satisfaction, S_i, of the ith objective:

$$S_1 = \alpha_1 \ (TDIST - I_1), \tag{5.6}$$

$$S_2 = \alpha_2 \ (LDDEV - I_2), \tag{5.7}$$

and $$S_3 = \alpha_3 \ (LNDEV - I_3). \tag{5.8}$$

The objective having the greatest value of S_i is selected as the least satisfactory achievement level.

To determine which tradeoff to accept, given that the least satisfactory achievement level has not been sufficiently improved in the last iteration, the potential improvement in the utility function, ΔU, is evaluated for each tradeoff presented to the decision maker. Thus, for the total distance objective,

$$\Delta U = \alpha_1 T_1 - \alpha_2 T_2 - \alpha_3 T_3, \tag{5.9}$$

where T_i is the tradeoff quantity displayed for the ith objective function. Any negative tradeoff quantity is not used in the calculation, since only positive achievement level relaxations are used in the subsequent single-objective problem, as explained in 'Goal Tradeoffs'.

Table 5.2 contains the results of fifteen problems using five different utility functions. For each utility function, three different starting points are used to obtain a final solution. The last column shows the percent error, measured against the minimum value obtained for the utility function. No two final solutions in the table are the same. However, the final utility value of most problems in the table are relatively close to one another. The major exception is problem twelve, which showed almost no improvement in the initial utility value, and which had a final solution 11.64 percent higher than the best final solution for the given utility function. The average error for the problems in the table is 3.46 percent.

TABLE 5.2

CONVERGENCE OF INTERACTIVE COMPUTER PROGRAM USING DIFFERENT INITIAL POINTS

Problem Number	Number of Cities	Utility Function			Ideal Solution			Ideal Utility	Initial Point			Initial Utility	Final Solution			Final Utility	% Error
		α_1	α_2	α_3	TDIST	LDDEV	LNDEV		TDIST	LDDEV	LNDEV		TDIST	LDDEV	LNDEV		
1	32	0.291	0.026	0.683	809	130	9	275.68	809	1570	51	311.07	853	870	12	279.04	*
2	"	"	"	"	"	"	"	"	884	330	27	284.27	850	670	26	282.53	1.25
3	"	"	"	"	"	"	"	"	938	780	13	302.12	882	630	13	281.92	1.03
4	"	0.543	0.032	0.425	"	"	"	447.27	823	1500	43	513.16	828	420	47	483.02	*
5	"	"	"	"	"	"	"	"	875	400	22	497.28	834	600	51	493.74	2.22
6	"	"	"	"	"	"	"	"	824	2150	31	529.41	826	670	49	490.78	1.61
7	50	0.029	0.893	0.078	548	1	1	16.86	633	11	12	29.12	568	3	28	21.34	*
8	"	"	"	"	"	"	"	"	762	8	1	29.32	619	3	19	22.11	3.61
9	"	"	"	"	"	"	"	"	667	4	69	28.30	666	3	14	23.09	8.20
10	"	0.059	0.914	0.027	"	"	"	33.27	584	9	36	43.65	559	3	19	36.24	*
11	"	"	"	"	"	"	"	"	591	6	10	40.62	581	2	18	36.59	0.97
12	"	"	"	"	"	"	"	"	612	6	2	41.65	606	5	5	40.46	11.64
13	75	0.151	0.443	0.406	851	3	11	134.30	905	6	93	177.07	920	6	24	149.99	*
14	"	"	"	"	"	"	"	"	851	8	92	169.40	934	8	21	153.10	2.07
15	"	"	"	"	"	"	"	"	903	8	43	157.34	941	8	18	152.94	1.97

*Minimum value of utility function obtained from this starting point.

Nondominance Analysis. Another way of measuring the effectiveness of the procedure is to determine whether the final solution is a member of the nondominated set. Since each single-objective problem is solved heuristically, there is no guarantee that an optimal value for the single objective is reached. This carries over to the multiobjective case, in which there is no guarantee of nondominance.

One method of measuring nondominance is to consider all solutions which are obtained in the course of solving a WBVRP, comparing each solution against the others. As stated previously, any WBVRP can be expected to generate about six of these solutions. The first 200 such solutions obtained in the analyses of Chapter VI were taken as a data base for the nondominance study. Of these 200 solutions, 30 were found to be dominated. This translates into a 15 percent rate for dominated solutions. For most decision makers, this rate is probably acceptable. Of course, there is no way of knowing whether the dominating solutions were themselves dominated by other (undiscovered) solutions.

Efficiency of Program

The efficiency of the interactive program is measured by the time required to reach a final solution. The CPU time required for each of the single-objective functions was covered previously in Chapter IV. Although the interactive program includes a small amount of overhead for the main program and for the graphic display routines, this overhead is negligible. If the worst case for each single-objective algorithm for the problems of Chapter IV were experienced in solving a WBVRP, and if two of each of those single-objective problems were solved in the course of finding a final solution, then a total CPU time of over two minutes could be expected. However, the majority of problems solved during this research required well under one minute total CPU time.

Another measure of efficiency is the total (clock) time for the analyst to arrive at the final answer to a WBVRP. This is a function of the amount of effort required to choose the least satisfactory achievement level, and to determine which tradeoff to select. For most of the problems in this research, the total clock time was less than ten minutes. Admittedly, had the author been solving real-world problems having more realistic tradeoff considerations, the total clock time might very well have been greater.

Example Problem

An example is now presented to demonstrate the use of the interactive computer program. The problem has 33 customers, a vehicle capacity of 150 units, and a distance limit of 50 miles per route. Distances are Euclidean. The problem details are given in Appendix B.

Figure 5.3 shows the initial route set presented to the decision maker. Recall that this route set is the best minimum distance solution chosen from eight successive runs of the Clarke and Wright savings algorithm, the 2-opt algorithm, and the 3-opt algorithm. This solution has a total distance of 174 miles, a route-load deviation of 27 units, and a route-length deviation of 22 miles. The decision maker is asked whether this solution is acceptable as a final solution. Since the driver of the longest route must travel more than twice the distance of the shortest route, the route set is not acceptable as a final solution. The decision maker selects route-length deviation as the objective to minimize.

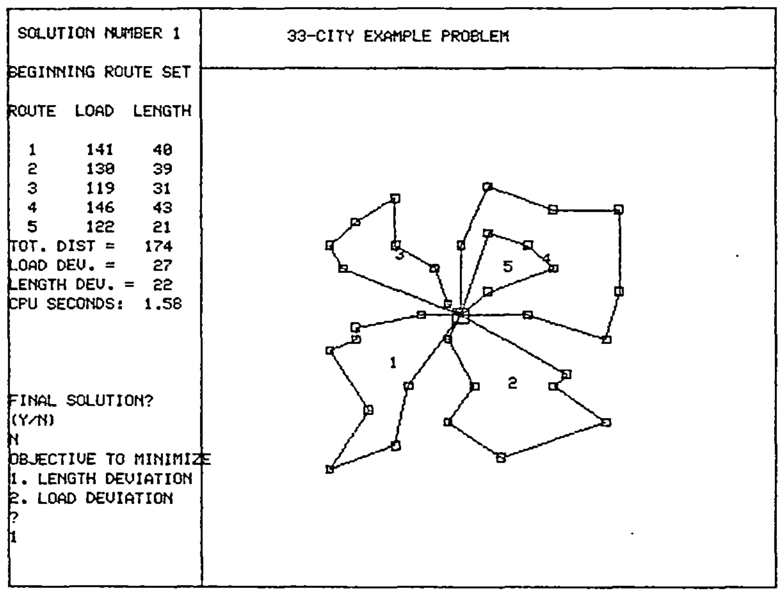
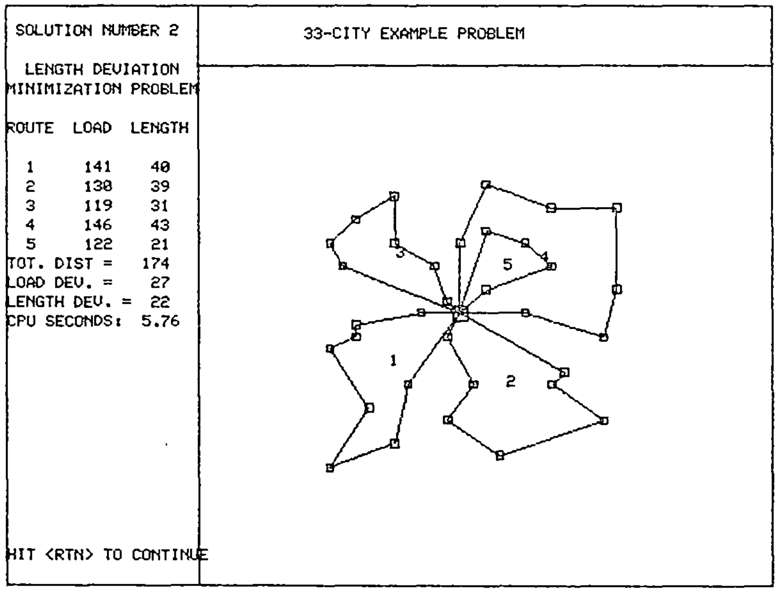


Figure 5.3. Example Problem: Beginning Route Set

Figure 5.4(A) shows the results of the route-length deviation minimization problem (solution number 2). This problem was solved using the previous values of total distance (174 miles) and route-load deviation (27 units) as constraints, and was solved in an attempt to drive the solution to the nondominated set. The solution to this problem is the same as the previous one, which indicates that it probably lies in the nondominated set.

Figure 5.4(B) contains the menu, the problem status, and a set of tradeoffs resulting from solution number two. Since no progress was made in reducing route-length deviation, the achievement level of 22 miles is considered the least satisfactory achievement level, and the third menu item (minimize length deviation) is selected for the next iteration.

Since the objective function for the next iteration is the same as in the current iteration, some relaxation in at least one of the other two achievement levels is necessary. The tradeoff information shown in Figure 5.4(B) can be used to determine the amount of constraint relaxation to allow for total distance and route-load deviation. The fourth tradeoff indicates that an improvement of at least 13 miles in route-length deviation can be obtained if total distance is increased by 5 miles. This tradeoff also shows that the immediate result of the arc exchange associated with the tradeoff will be a decrease (indicated by the negative sign) of 5 units in route-load deviation. However, this negative value should not be used in determining which tradeoff to use, unless the competing tradeoffs are otherwise equivalent. The problem resulting from the tradeoff will actually have zero constraint relaxation for route-load deviation, as explained previously in 'Goal Tradeoffs'.



(A) ROUTE SET DISPLAY

```
                    33-CITY EXAMPLE PROBLEM
                     SOLUTION NUMBER  2

           MAIN MENU              STATUS  LIMIT        ROUTES:
                                                     # LOAD DIST
  1. MINIMIZE TOTAL DISTANCE    --  174    174       1  141   40
  2. MINIMIZE LOAD DEVIATION    --   27     27       2  130   39
  3. MINIMIZE LENGTH DEVIATION  --   22   ****       3  119   31
  4. MANUAL ROUTE IMPROVEMENT                        4  146   43
  5. DISPLAY PREVIOUS SOLUTION                       5  122   21
  6. BACKTRACK TO PREV. SOL.
  7. REMOVE ROUTE FROM CALC.
  8. EXIT
                      ESTIMATED TRADEOFFS:
                                   1    2    3    4    5    6
    LENGTH DEVIATION IMPROVEMENT  19   17   16   13    4    2
    TOTAL DISTANCE    RELAXATION  28   19   13    5    4    2
    LOAD DEVIATION    RELAXATION -11   -7   17   -5    4   -5
    ORIGINAL TRADEOFFS 458    REDUCED TRADEOFFS   7
  SELECT FROM MENU
3
WHICH TRADEOFF # IS ACCEPTABLE?
?
4
```

(B) INTERACTIVE SCREEN DISPLAY

Figure 5.4. Example Problem: Solution Number 2

Having decided that an improvement of 13 miles in route-length deviation for an increase of 5 miles in total distance is an attractive tradeoff, the decision maker selects tradeoff number four, as shown in Figure 5.4(B). The problem to be solved in the next iteration is

$$\text{Min LNDEV} \tag{5.10}$$

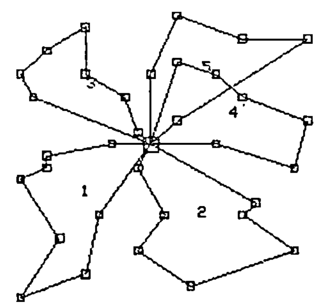
$$\text{S.T. TDIST} \leqslant 179 \tag{5.11}$$

$$\text{and LDDEV} \leqslant 27. \tag{5.12}$$

Had the decision maker desired, a route-length deviation problem could have been solved without choosing one of the displayed tradeoffs. Entering a zero instead of a valid tradeoff number would result in a request for new constraints on total distance and route-load deviation. With these constraints, a route-length deviation problem would then be solved. This method generally yields inferior results to the tradeoff method, however, and can actually result in a dominated tradeoff which had been removed from the original tradeoff set. The use of one of the displayed tradeoffs guarantees at least the amount of improvement shown in the tradeoff, since the first thing the computer does is to perform the arc exchange associated with the tradeoff, before solving problem (5.10) - (5.12).

The solution to problem (5.10) - (5.12) is shown in Figure 5.5 (solution number 3). The route-length deviation algorithm has reduced the value of LNDEV to 9 miles, while TDIST has increased to 179 miles and LDDEV has increased to 27 units. This is the result tradeoff number four had indicated. After the tradeoff was made, no further improvement in the objective function could be made. The 'improvement ratio' for this tradeoff is unity.

```
SOLUTION NUMBER 3                 33-CITY EXAMPLE PROBLEM

 LENGTH DEVIATION
MINIMIZATION PROBLEM

ROUTE  LOAD  LENGTH

 1      141    40
 2      130    39
 3      119    31
 4      137    33
 5      131    36
TOT. DIST =   179
LOAD DEV. =    22
LENGTH DEV. =   9
CPU SECONDS: 11.15
```




```
HIT <RTN> TO CONTINUE
```

(A) ROUTE SET DISPLAY

```
                      33-CITY EXAMPLE PROBLEM
                       SOLUTION NUMBER  3

            MAIN MENU                  STATUS  LIMIT         ROUTES:
                                                          #  LOAD DIST
     1. MINIMIZE TOTAL DISTANCE    —    179     179       1  141   40
     2. MINIMIZE LOAD DEVIATION    —     22      27       2  130   39
     3. MINIMIZE LENGTH DEVIATION  —      9    ****       3  119   31
     4. MANUAL ROUTE IMPROVEMENT                          4  137   33
     5. DISPLAY PREVIOUS SOLUTION                         5  131   36
     6. BACKTRACK TO PREV. SOL.
     7. REMOVE ROUTE FROM CALC.
     8. EXIT
                      ESTIMATED TRADEOFFS:
                                   1    2    3    4
      LENGTH DEVIATION IMPROVEMENT  7    5    4    3
      TOTAL DISTANCE   RELAXATION  18   14   10    2
      LOAD DEVIATION   RELAXATION  29  -17    6   -4
      ORIGINAL TRADEOFFS 166    REDUCED TRADEOFFS   4
     SELECT FROM MENU
2
SPECIFY NEW LIMITS FOR LOAD DEVIATION   PROBLEM
TOTAL DISTANCE
?
185
LENGTH DEVIATION
?
10
```

(B) INTERACTIVE SCREEN DISPLAY

Figure 5.5. Example Problem: Solution Number 3

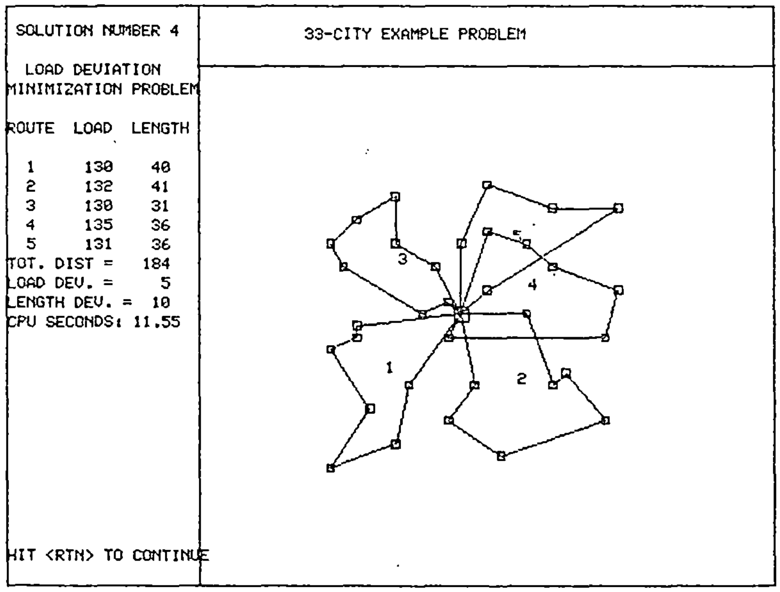
The decision maker must now choose the least satisfactory achievement level from the three values shown in Figure 5.5; i.e., total distance (179 miles), route-load deviation (22 units), or route-length deviation (9 miles). Suppose that route-load deviation is selected (menu option two). This response is shown in Figure 5.5(B). Since the new objective function is different from the previous one, the tradeoffs shown in Figure 5.5(B) will not be utilized. Instead, the decision maker is asked for new limits on the two new constraints, total distance and route-length deviation. The decision maker enters these limits as 185 miles and 10 miles, respectively. The new problem to be solved is

$$\text{Min LDDEV} \tag{5.13}$$

$$\text{S.T. TDIST} \leq 185 \tag{5.14}$$

$$\text{and LNDEV} \leq 10. \tag{5.15}$$

Figure 5.6 shows the solution to problem (5.13) -(5.15). Route-load deviation has been reduced to 5 units, while total distance is now 184 miles and route-length deviation is 10 miles. Once again, the decision maker must decide which of the three achievement levels is least satisfactory. Suppose that route-length deviation is selected. The objective function has changed, so the single tradeoff displayed in Figure 5.6(B) is not used. Instead the decision maker is asked for new limits on the two constraints, total distance and route-load deviation. At this point, the decision maker is willing to relax the constraints very little. The new limits are 185 miles and 5 units, respectively. The new problem to be solved is:



(A) ROUTE SET DISPLAY

```
                     33-CITY EXAMPLE PROBLEM
                      SOLUTION NUMBER  4

          MAIN MENU                 STATUS  LIMIT         ROUTES:
                                                        # LOAD DIST
   1. MINIMIZE TOTAL DISTANCE    --   184    185        1 130   40
   2. MINIMIZE LOAD DEVIATION    --     5   ****        2 132   41
   3. MINIMIZE LENGTH DEVIATION  --    10     10        3 130   31
   4. MANUAL ROUTE IMPROVEMENT                          4 135   36
   5. DISPLAY PREVIOUS SOLUTION                         5 131   36
   6. BACKTRACK TO PREV. SOL.
   7. REMOVE ROUTE FROM CALC.
   8. EXIT
                     ESTIMATED TRADEOFFS:
                                 1
     LOAD DEVIATION   IMPROVEMENT     1
     TOTAL DISTANCE   RELAXATION      9
     LENGTH DEVIATION RELAXATION      4
     ORIGINAL TRADEOFFS   3    REDUCED TRADEOFFS    1
   SELECT FROM MENU
3
SPECIFY NEW LIMITS FOR LENGTH DEVIATION PROBLEM
TOTAL DISTANCE
?
185
LOAD DEVIATION
?
5
```

(B) INTERACTIVE SCREEN DISPLAY

Figure 5.6. Example Problem: Solution Number 4

$$\text{Min LNDEV} \tag{5.16}$$

$$\text{S.T. TDIST} \leq 185 \tag{5.17}$$

$$\text{and LDDEV} \leq 5. \tag{5.18}$$

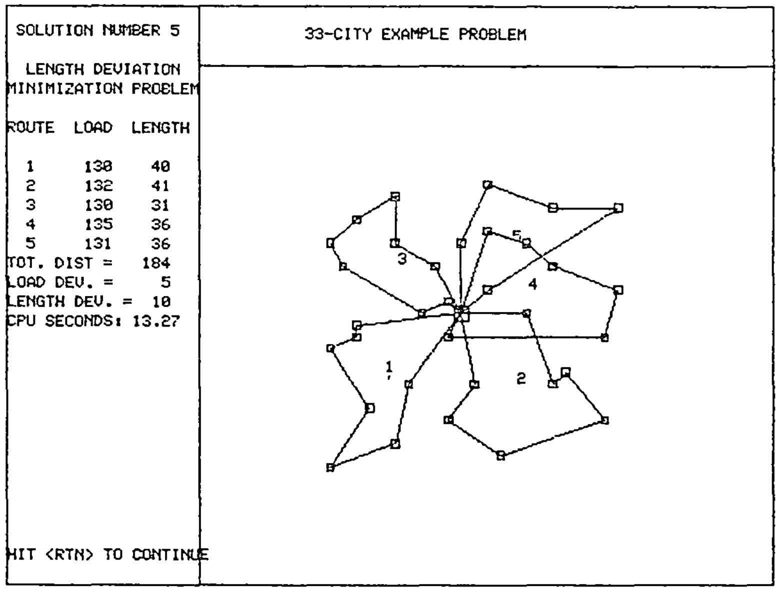
The solution to problem (5.16) - (5.18) is shown in Figure 5.7. The route-length deviation has not been decreased, due to the very slight relaxation (one mile) the decision maker allowed in the total distance constraint. However, the decision maker is now presented with a set of tradeoffs (Figure 5.7(B)) which will allow a better evaluation of the effects of constraint relaxation. In fact, allowing no constraint relaxation at all and solving a problem just to obtain such a set of tradeoffs is an acceptable practice using the interactive program. The second tradeoff indicates that route-length deviation can be reduced by three miles if total distance is increased by two miles and route-load deviation is increased by four units. The decision maker selects this tradeoff, as indicated by the response in Figure 5.7(B). The new problem to be solved is

$$\text{Min LNDEV} \tag{5.19}$$

$$\text{S.T. TDIST} \leq 186 \tag{5.20}$$

$$\text{and LDDEV} \leq 9. \tag{5.21}$$

The solution to problem (5.19) - (5.21) is shown in Figure 5.8. The total distance of the set of routes shown in Figure 5.8(A) is 186 miles, route-load deviation is 9 units, and route-length deviation is 7 miles. At this point, the decision maker is unable to select one of these achievement levels as least satisfactory, so the procedure is halted by selecting menu option number eight. Comparing this final solution to the initial (minimum distance) solution, it is seen that



(A) ROUTE SET DISPLAY

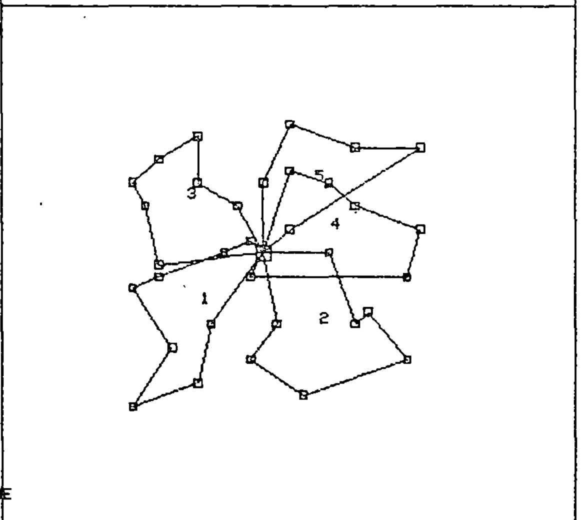
```
                    33-CITY EXAMPLE PROBLEM
                     SOLUTION NUMBER  5

            MAIN MENU               STATUS  LIMIT        ROUTES:
                                                       # LOAD DIST
   1. MINIMIZE TOTAL DISTANCE     --   184    185      1  130   40
   2. MINIMIZE LOAD DEVIATION     --     5      5      2  132   41
   3. MINIMIZE LENGTH DEVIATION   --    10   ****      3  130   31
   4. MANUAL ROUTE IMPROVEMENT                         4  135   36
   5. DISPLAY PREVIOUS SOLUTION                        5  131   36
   6. BACKTRACK TO PREV. SOL.
   7. REMOVE ROUTE FROM CALC.
   8. EXIT
                      ESTIMATED TRADEOFFS:
                                   1    2    3    4
     LENGTH DEVIATION IMPROVEMENT   6    3    2    1
     TOTAL DISTANCE    RELAXATION   9    2   -1   -5
     LOAD DEVIATION    RELAXATION   1    4   15    2
     ORIGINAL TRADEOFFS  58    REDUCED TRADEOFFS   4
   SELECT FROM MENU
3
WHICH TRADEOFF # IS ACCEPTABLE?
?
2
```

(B) INTERACTIVE SCREEN DISPLAY

Figure 5.7. Example Problem: Solution Number 5

```
SOLUTION NUMBER 6

 LENGTH DEVIATION
MINIMIZATION PROBLEM

ROUTE  LOAD  LENGTH

  1     126    39
  2     132    41
  3     134    34
  4     135    36
  5     131    36
TOT. DIST =   186
LOAD DEV. =     9
LENGTH DEV. =   7
CPU SECONDS: 15.99

HIT <RTN> TO CONTINUE
```

33-CITY EXAMPLE PROBLEM



(A) ROUTE SET DISPLAY

```
                         33-CITY EXAMPLE PROBLEM
                          SOLUTION NUMBER  6

            MAIN MENU                   STATUS  LIMIT        ROUTES:
                                                           #  LOAD DIST
   1. MINIMIZE TOTAL DISTANCE      —     186     186       1  126    39
   2. MINIMIZE LOAD DEVIATION      —       9       9       2  132    41
   3. MINIMIZE LENGTH DEVIATION    —       7    ****       3  134    34
   4. MANUAL ROUTE IMPROVEMENT                             4  135    36
   5. DISPLAY PREVIOUS SOLUTION                            5  131    36
   6. BACKTRACK TO PREV. SOL.
   7. REMOVE ROUTE FROM CALC.
   8. EXIT
                      ESTIMATED TRADEOFFS:
                                      1    2
    LENGTH DEVIATION IMPROVEMENT      2    1
    TOTAL DISTANCE   RELAXATION       8   -5
    LOAD DEVIATION   RELAXATION       5    2
    ORIGINAL TRADEOFFS  31    REDUCED TRADEOFFS    2
   SELECT FROM MENU
8

READY
```

(B) INTERACTIVE SCREEN DISPLAY

Figure 5.8. Example Problem: Solution Number 6

route-load deviation has been reduced by 66.67 percent and route-length deviation has been reduced by 68.18 percent. These improvements in workload balance were made at a penalty of 6.90 percent in total distance.

Summary

In this chapter, the single-objective algorithms of Chapter IV have been consolidated into an interactive computer program to enable the decision maker to solve the WBVRP using a heuristic version of the Method of Satisfactory Goals. Because dual variables are not available, the program provides the user with tradeoffs which determine the amount of constraint relaxation at a given iteration of the procedure. The tradeoffs were evaluated in terms of their ability to predict improvement in an objective function using a specified level of constraint relaxation. The effectiveness of the interactive program was evaluated in terms of its ability to converge to a solution from different starting points, and by the percentage of dominated solutions generated by the procedure. The efficiency of the program was evaluated in terms of the amount of time (both CPU and elapsed) required to reach a final solution. Finally, the use of the interactive computer program was demonstrated by solving a sample WBVRP.

CHAPTER VI

WORKLOAD BALANCING COSTS

Introduction

A set of workload balanced routes can be expected in most cases to be more costly in terms of the total distance driven. This extra distance will translate to added fuel and maintenance costs for the fleet and, depending on the particular distribution system, might translate to added driver and/or helper costs. In the sample problem of Chapter V, the extra distance was 6.90 percent. Other costs, such as administration, dispatching, hardware and software, might or might not increase. These other costs are not addressed in this chapter.

The cost penalty for workload balancing will depend on the extent to which the routes are to be balanced. In addition, the depot location, customer demand pattern, and spatial characteristics (customer location pattern) of the problem can be expected to affect the penalty. In this chapter, it is assumed that routes are to be balanced as much as possible, allowing the analysis to concentrate on the effects of depot location and demand and spatial characteristics.

Attention is focused on distribution systems having unbalanced workloads. It is desired to know what penalty must be paid in going from unbalanced, minimum distance routes to routes which are balanced in one or both of the workload elements, and the effect that the problem characteristics have on the penalty. The workload balancing penalty is defined as the fraction of total route distance which must be increased in order to balance the workload element(s).

It is easy to imagine situations in which route balancing is affected by having some degree of variability in the workload element being balanced. Consider a route-load balancing problem in which demand is constant and two route loads differ by a given amount, say, twice the constant demand. Since all demands are the same, no simple pairwise exchange can balance the loads. However, if the demand varies from customer to customer, there is a chance that fewer exchanges will balance the loads and that the resulting distance penalty will be less. Of course, depending on circumstances, the opposite effect could result, requiring more exchanges and a greater distance penalty. And, if only one workload element is being balanced, the degree of variability in the workload element not being balanced can affect the total distance penalty. In the present example, variation in the distances between customers could result in the total distance penalty being greater or less than would result under a uniform spatial pattern.

The purpose of the analysis, then, is to determine whether any conclusions can be made regarding the effect that a problem's characteristics have on the workload balancing penalty. The analysis is performed for route load balancing, route length balancing, and balancing of both route loads and route lengths.

Method of Analysis

Two different approaches could be taken in analyzing the effects of depot location, customer demand pattern, and customer spatial pattern on the workload balancing penalty. In the first approach, a large random sample of problems having different characteristics could be solved, followed by a statistical analysis of the results. In the second, a 'standard' problem could be set up, then solved several times as different characteristics are systematically varied while holding other characteristics constant. The second approach was taken in this analysis, primarily because it was felt that more insight could be gained by thoroughly studying the standard problem, but also because of the amount of effort which would be involved in creating and checking out each new randomly generated problem before solving it. It is realized that any conclusions reached by this type of analysis will not necessarily apply to all problems. However, the results might lead to hypotheses which can be tested through further research.

The standard problem used in the analysis contains 36 customers served by four vehicles having a capacity of 110 units each and no limit on route length. The customers have an average demand of approximately ten units each. Distances are Euclidean. To study the effect of demand variability, four different demand patterns are used. The first pattern has a constant demand of ten units with no variability. The second pattern has an average demand of 9.78 and a standard deviation of 1.51, and was created by randomly generating integer values between 8 and 12, inclusive, from a uniform distribution. The third pattern has an average demand of 9.78 and a standard deviation of 2.12, created by randomly generating values between 7 and 13, inclusive, from a uniform distribution. Finally, a fourth demand pattern has an average of 9.83 and a standard deviation of 3.73, having integer values between 4 and 16, inclusive. Each demand in the four patterns is shown in Table 6.1.

To study the effect of spatial characteristics, a six-by-six grid

TABLE 6.1

CUSTOMER DEMAND PATTERNS

Customer	Demand Pattern 1	Demand Pattern 2	Demand Pattern 3	Demand Pattern 4
1	10	8	13	4
2	10	9	9	10
3	10	10	10	15
4	10	12	9	14
5	10	8	11	5
6	10	10	11	15
7	10	8	7	6
8	10	9	13	13
9	10	9	7	13
10	10	12	12	13
11	10	11	8	10
12	10	8	10	5
13	10	11	10	8
14	10	8	7	16
15	10	12	8	6
16	10	11	11	5
17	10	9	7	12
18	10	8	13	11
19	10	11	8	10
20	10	8	11	7
21	10	12	8	12
22	10	9	7	13
23	10	8	12	8
24	10	12	12	13
25	10	11	10	9
26	10	8	11	5
27	10	8	8	8
28	10	9	13	12
29	10	12	8	12
30	10	10	7	14
31	10	12	12	12
32	10	10	9	4
33	10	10	12	4
34	10	8	7	6
35	10	10	8	16
36	10	11	13	8

pattern was established and customers were located on the grid. The grid points are separated by ten distance units in the vertical and horizontal directions. Four different spatial patterns were then established. The first spatial pattern has each of the 36 customers located on one of the grid points. The second pattern has each customer randomly placed within plus or minus three distance units in the vertical and horizontal directions from a grid point. The third and fourth patterns have customers randomly placed within plus or minus five distance units and within plus or minus ten distance units from the grid points, respectively. The four different customer location patterns are shown in Figures 6.1 through 6.4. An examination of these figures shows an increase in the variability of distances between customers with the first through the fourth pattern, respectively. One measure which could be used to quantify the spatial characteristic is the standard deviation of distances between nearest neighbors divided by the average distance between nearest neighbors. This measure is found to be 0.00, 0.16, 0.38, and 0.42 for the first through fourth customer location pattern, respectively. This measure of spatial dispersion may or may not be useful in predicting the workload balancing penalties. In either case, spatial pattern one is referred to as 'highly structured', pattern two is referred to as 'somewhat structured', and patterns three and four are referred to as 'unstructured'.

To study the effect of depot location on the workload balancing penalty, the depot was moved from the centroid to the outer bound of the location pattern in different problem steps. Then for each combination of demand pattern, spatial pattern, and depot location, four different problems were solved. The objective functions to be minimized in the

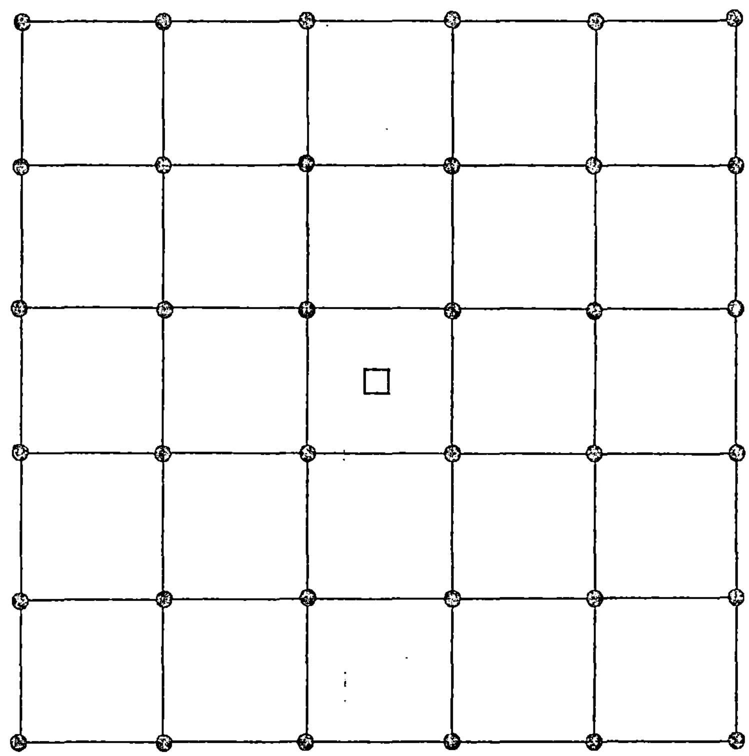


Figure 6.1. First Customer Location Pattern

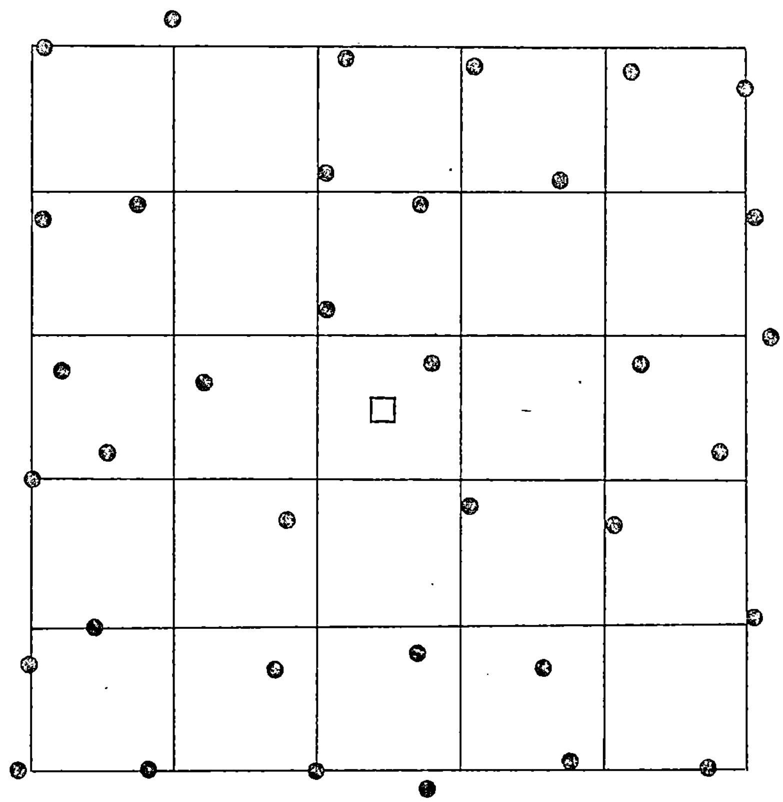


Figure 6.2. Second Customer Location Pattern

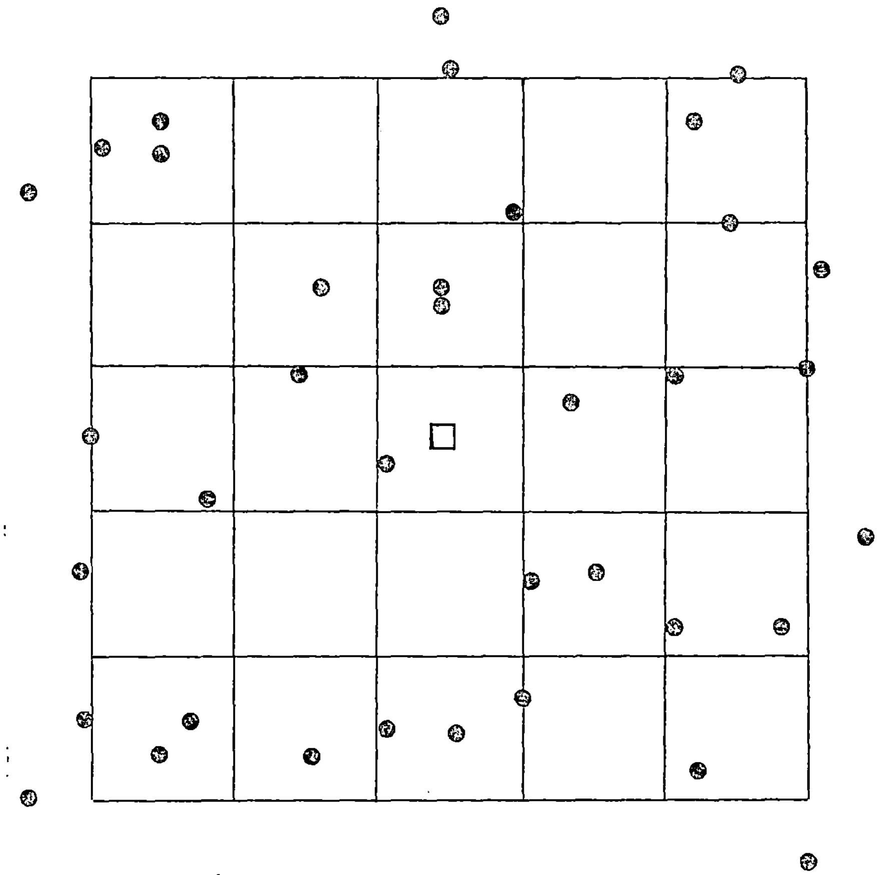


Figure 6.3. Third Customer Location Pattern

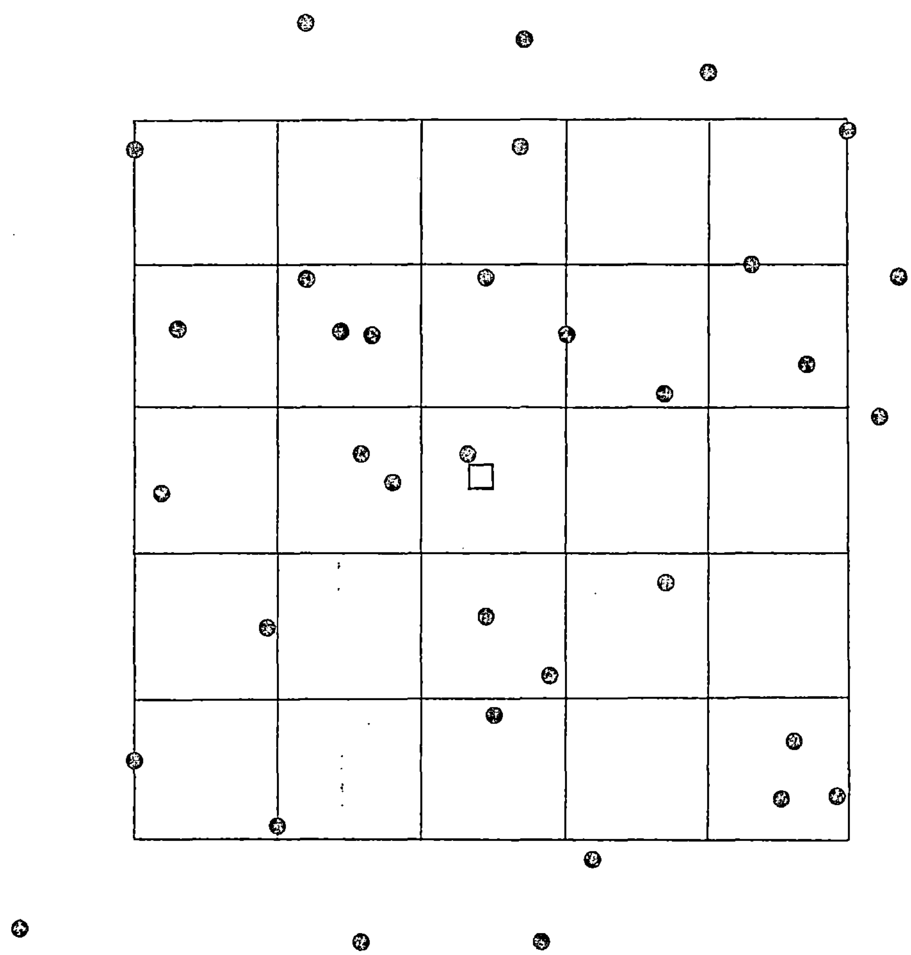


Figure 6.4. Fourth Customer Location Pattern

four problems were:

1. Total distance

2. Route-load deviation

3. Route-length deviation

4. Route-load and route-length deviation, equally weighted.

A total of 192 WBVRPs were solved using the interactive program of Chapter V.

Results

Depot Location Penalty

Before addressing the problem of workload balancing penalties, it is of interest to know the effect that depot location has on the costs of a distribution system that operates with minimum total distance routes. If the total distance of the routes increases as the depot is located away from the centroid of customer locations, then a 'depot location penalty' is paid by that distribution system. For a particular combination of demand pattern, spatial pattern, and depot location, the depot location penalty is calculated as

$$P_{Loc} = (TDIST_{Min} - TDISTC_{Min})/TDISTC_{Min} \quad (6.12)$$

where P_{Loc} = depot location penalty,

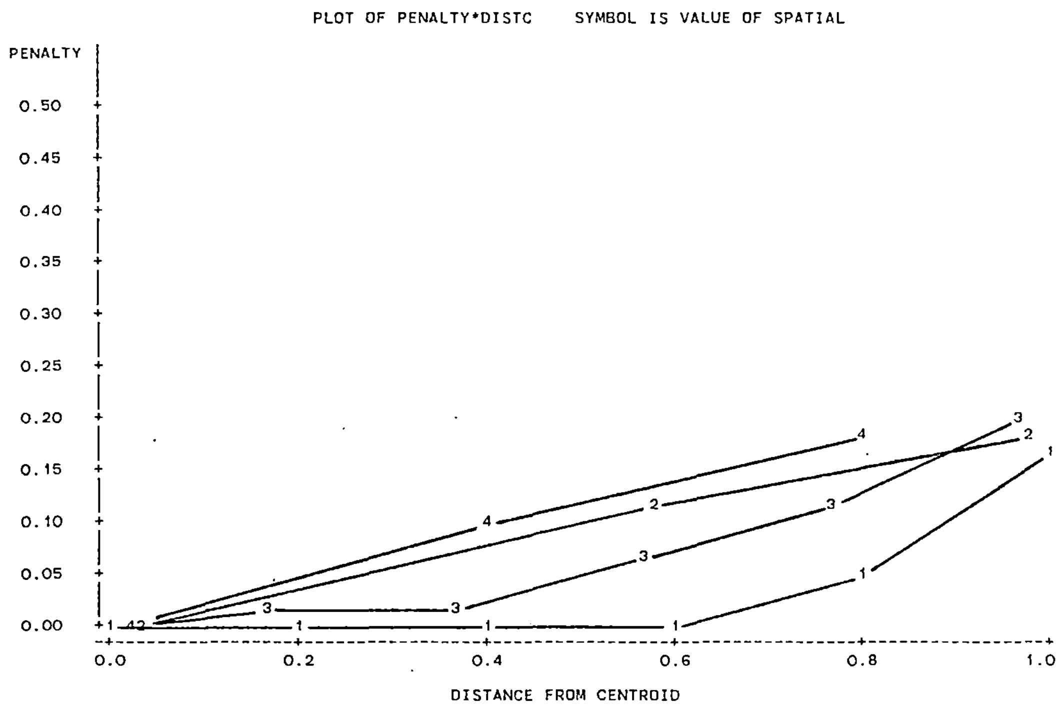
$TDIST_{Min}$ = total distance obtained in distance minimization problem at the given depot location,

and $TDISTC_{Min}$ = total distance obtained in distance minimization problem at the centroid.

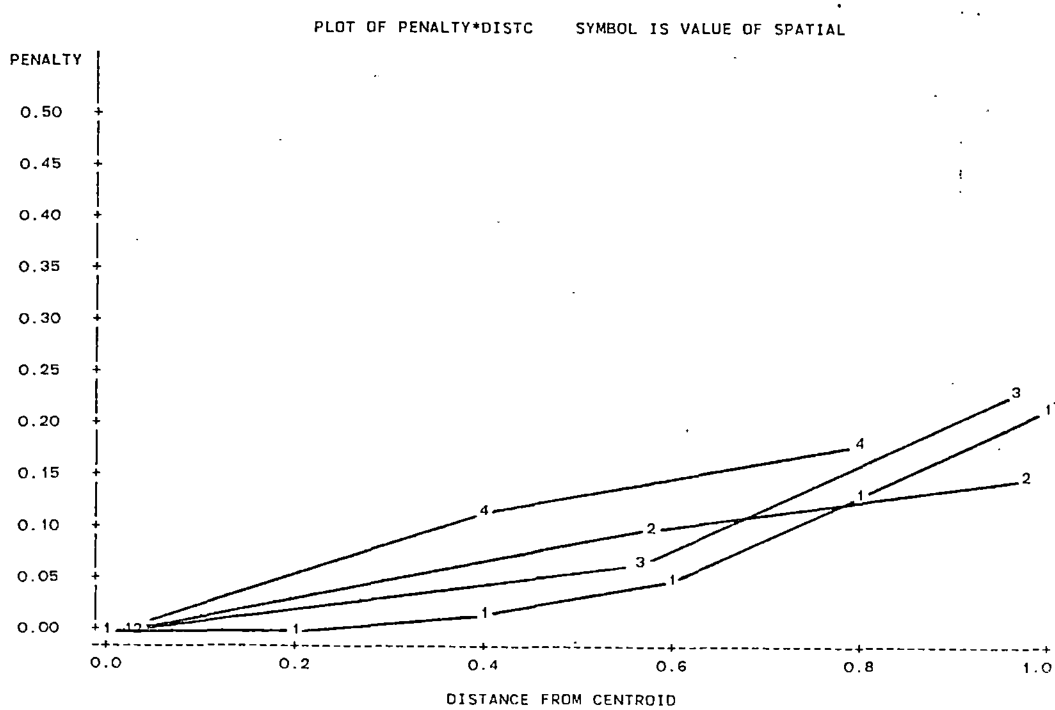
The depot location penalties are plotted in Figure 6.5 as a function of the depot distance from the centroid (expressed as a fraction of the distance from the centroid to the grid boundary), demand pattern, and spatial pattern. The penalty is seen to increase as the depot moves away from the centroid. This can be explained by the fact that routes formed when the depot is located at the centroid tend to be non-overlapping; but as the depot is moved away from the centroid, the routes must intersect in order to maintain vehicle capacity constraints. This intersecting of the routes causes the total distance, and thus the penalty, to be increased. The lowest penalties are obtained when demand is constant and the spatial pattern is highly structured (pattern number one). Any deviation from this combination causes the penalties to worsen.

For the most part, the depot location penalties are related to the spatial patterns, but not as a function of the simple measure of spatial dispersion defined above. In fact, the penalties for spatial patterns two and four are similar, yet the two patterns are totally different, pattern two being somewhat structured and pattern four being unstructured. This indicates that the simple measure of spatial dispersion is inadequate to predict the depot location penalties.

The effect of demand variability on the depot location penalty can be seen in Figure 6.5. For a very structured spatial pattern, the penalty is greatly increased as demand variability is increased. For the other spatial patterns, the penalty is decreased for some demand patterns, increased for others. The overall effect of increased variability of demand is to decrease the spread between the highest and lowest penalty values, lessening the effect of spatial pattern. It is likely that the effect of spatial pattern would be shown to be even less if demand variability were increased beyond that of demand pattern four, eventually taking on values close to those shown for the highly structured spatial pattern (number one) in Figure 6.5.

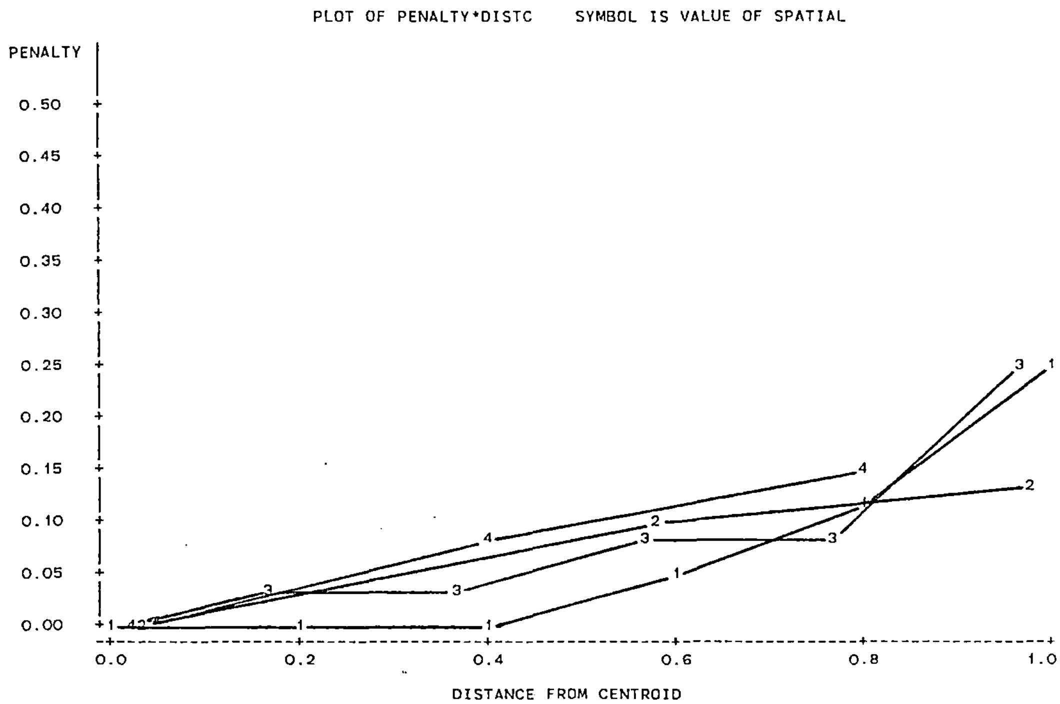


(A) FIRST DEMAND PATTERN

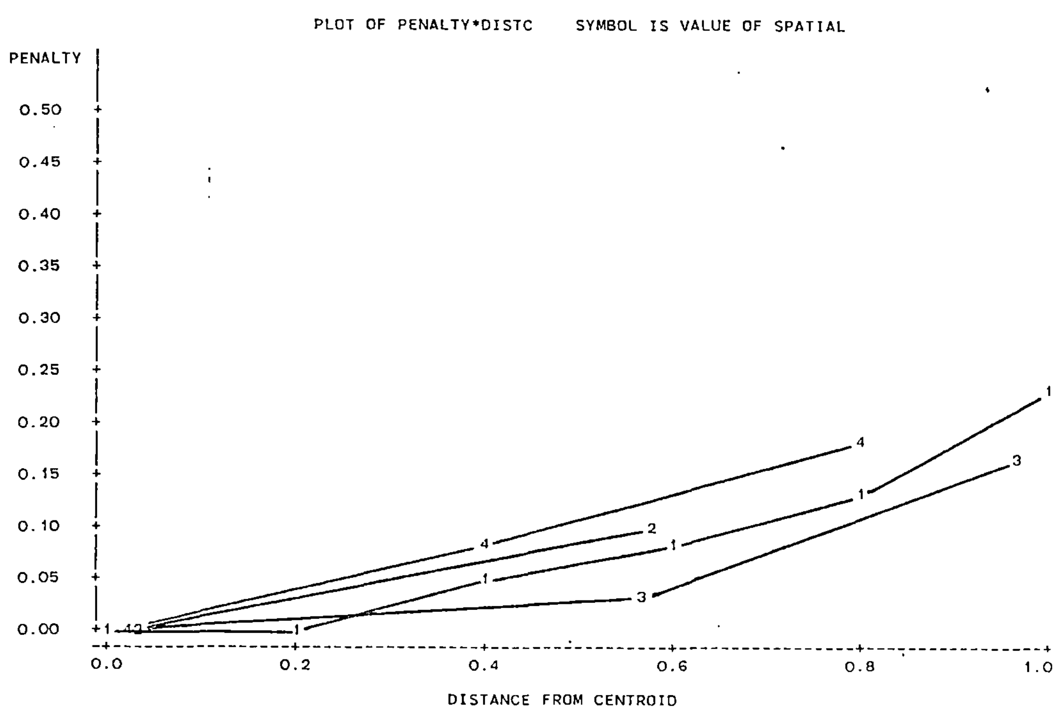


(B) SECOND DEMAND PATTERN

Figure 6.5. Depot Location Penalties for Minimum-Distance Problems



(C) THIRD DEMAND PATTERN



(D) FOURTH DEMAND PATTERN

Figure 6.5. (Continued)

To be competitive, the distribution system should have its depot as close to the centroid as possible. The depot location penalties for minimum distance routes can be thought of as 'base case' penalties, to which are added the workload balancing penalties of the next section. For instance, if a 0.10 depot location penalty is considered the highest that could be tolerated, then the depot could be located as far away as 80 percent of the distance to the grid boundary or no farther away than 40 percent, depending on the particular combination of demand and spatial patterns. However, if the depot is located at the 0.10 penalty limit, workload balancing would not be an attractive option because the distribution system could not pay the added penalty for workload balancing and remain competitive. For this reason, in the next section it will be assumed that the depot is not located farther away than 60 percent of the distance, an average of the two extremes.

Workload Balancing Penalties

For a particular combination of demand pattern, spatial pattern, and depot location, the workload balancing penalties are calculated as follows:

$$P_{LDDEV} = (TDIST_{LDDEV} - TDIST_{Min})/TDIST_{Min}, \tag{6.2}$$

$$P_{LNDEV} = (TDIST_{LNDEV} - TDIST_{Min})/TDIST_{Min}, \tag{6.3}$$

and

$$P_{BOTH} = (TDIST_{BOTH} - TDIST_{Min})/TDIST_{Min}, \tag{6.4}$$

where P_{LDDEV} = penalty for route-load balancing,

P_{LNDEV} = penalty for route-length balancing,

P_{BOTH} = penalty for balancing both route load and route length,

$TDIST_{LDDEV}$ = total distance obtained in route-load deviation problem,

$TDIST_{LNDEV}$ = total distance obtained in route-length deviation problem,

$TDIST_{BOTH}$ = total distance obtained in route-load and route-length deviation problem,

and $TDIST_{Min}$ = total distance obtained in distance minimization problem.

Route-load Balancing. The route-load balancing penalties are plotted in Figure 6.6 as a function of depot distance from the centroid, demand pattern, and spatial pattern. In Figure 6.6(A) it can be seen that, for the system with constant demand, route-load balancing is an attractive option regardless of the spatial pattern if distribution management is willing to pay a penalty of say, 0.10 over the minimum distance routing costs. With any demand variability at all, however, the penalty is a complex function of the distance from the centroid, demand variability, and spatial pattern. Only for a highly structured spatial pattern (number one) can the penalty be expected to be acceptable regardless of demand, and this spatial pattern is highly unlikely in real-world problems.

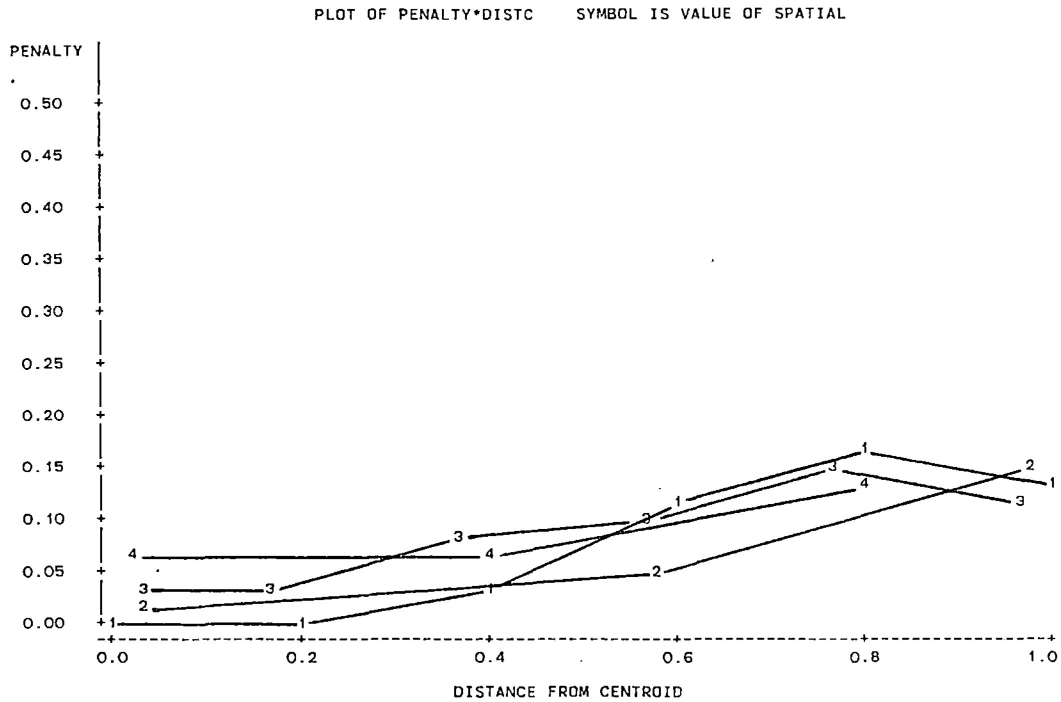
Route-length Balancing. The route-length balancing penalties are plotted in Figure 6.7 as a function of depot distance from the centroid, demand pattern, and spatial pattern. Here it can be seen that the penalty tends to be less for the structured location patterns (one and two) than for the unstructured patterns (three and four). Using a penalty limit of 0.10, the unstructured location patterns would not yield acceptable penalties except in a few cases, and then only if the depot were located near the centroid. The effect of demand variability is not apparent.

Route-load and Route-length Balancing. The penalties for balancing both route loads and route lengths are plotted in Figure 6.8. The plots are very similar to those of Figure 6.7, and the same general comments apply. Overall, the penalties are slightly higher than those of Figure 6.7, as could be expected. In some cases the penalty is lower, but this is because solutions were accepted in those cases which did not obtain the absolute minimum of both route balancing measures, but rather a very low value of each. The implication is that in those situations in which route lengths are balanced, the route loads could also be balanced if distribution management is willing to make minor tradeoffs between the two.

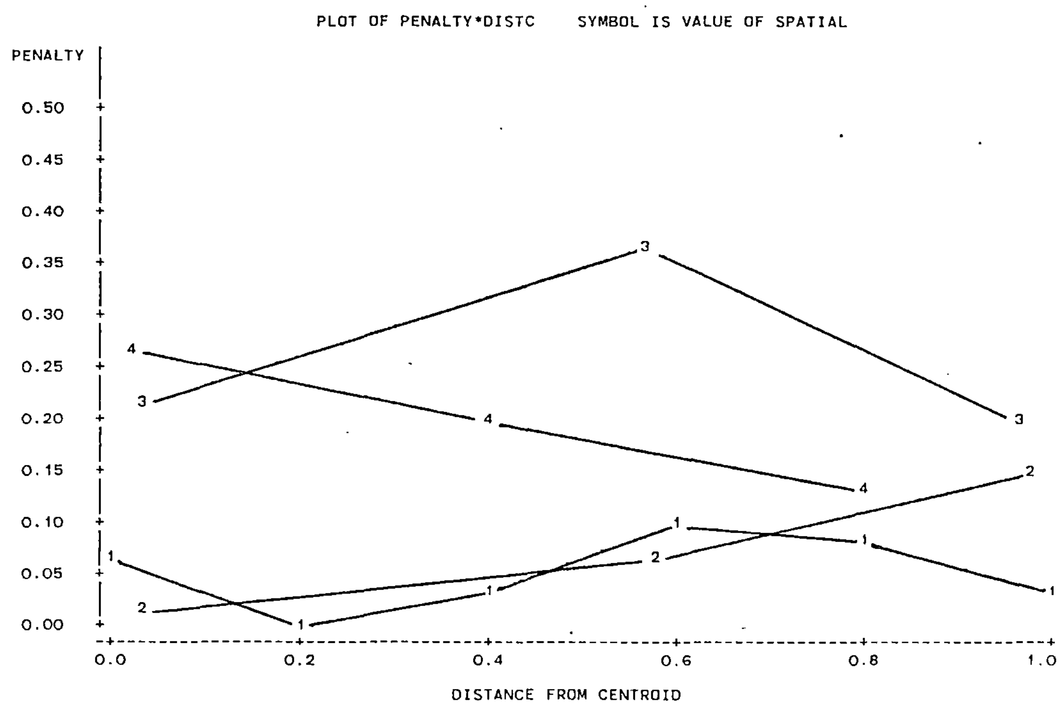
Conclusions

The results for the 36-city 'standard' problem were not as conclusive as had been hoped. In many cases the costs of workload balancing, and therefore the economic attractiveness of using workload-balanced routes, depend on a complex interaction of the problem variables. However, the following statements can be made:

1. For minimum-distance routes, the total route distance is least if the depot is situated near the center of all customer locations. Distribution systems having such centrally located depots are therefore more able to pay an additional cost for workload balancing than are those distrubution systems not having centrally located depots.
2. If demand is constant, balancing of route loads causes a relatively small penalty regardless of the spatial pattern of the customers. This should have implications not only for those

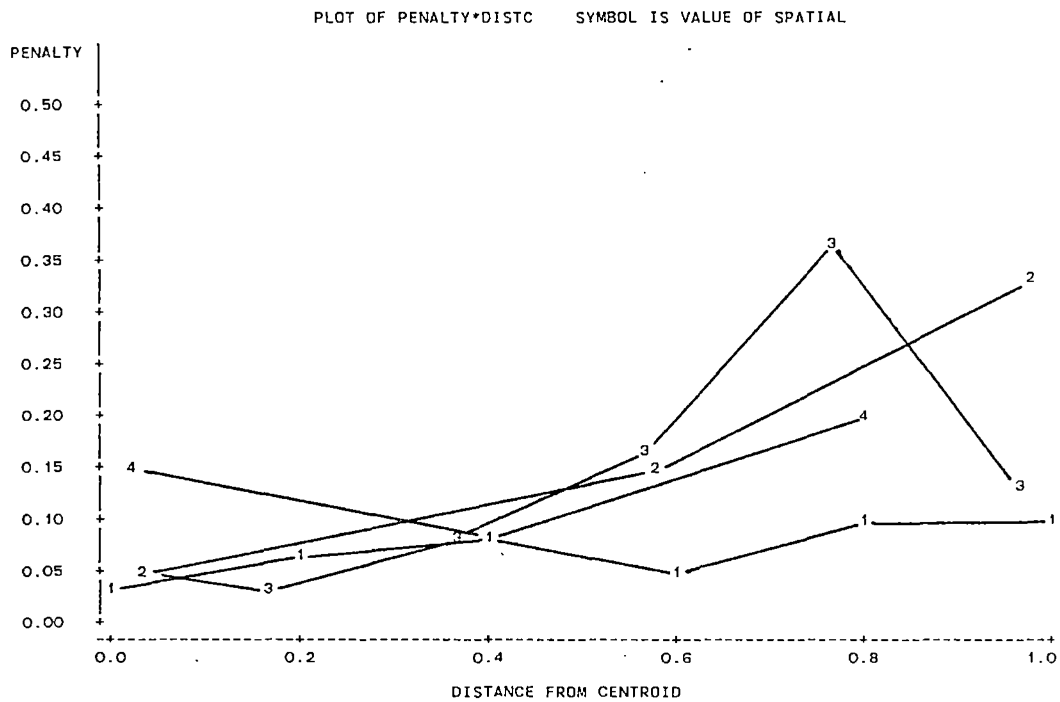


(A) FIRST DEMAND PATTERN

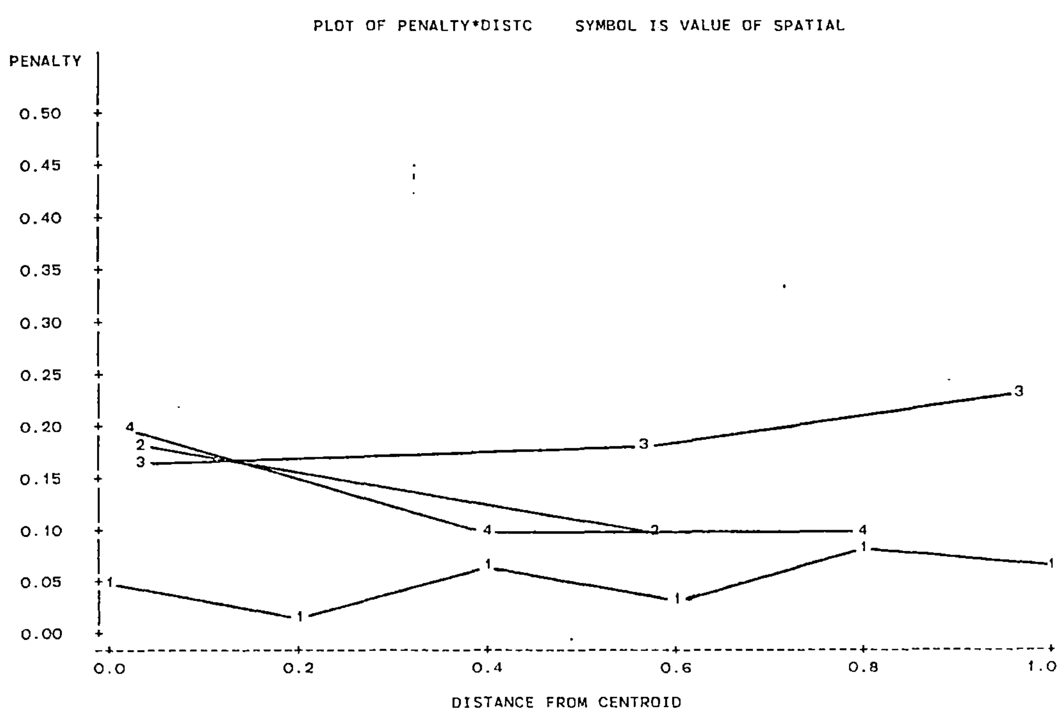


(B) SECOND DEMAND PATTERN

Figure 6.6. Route-Load Balancing Penalties

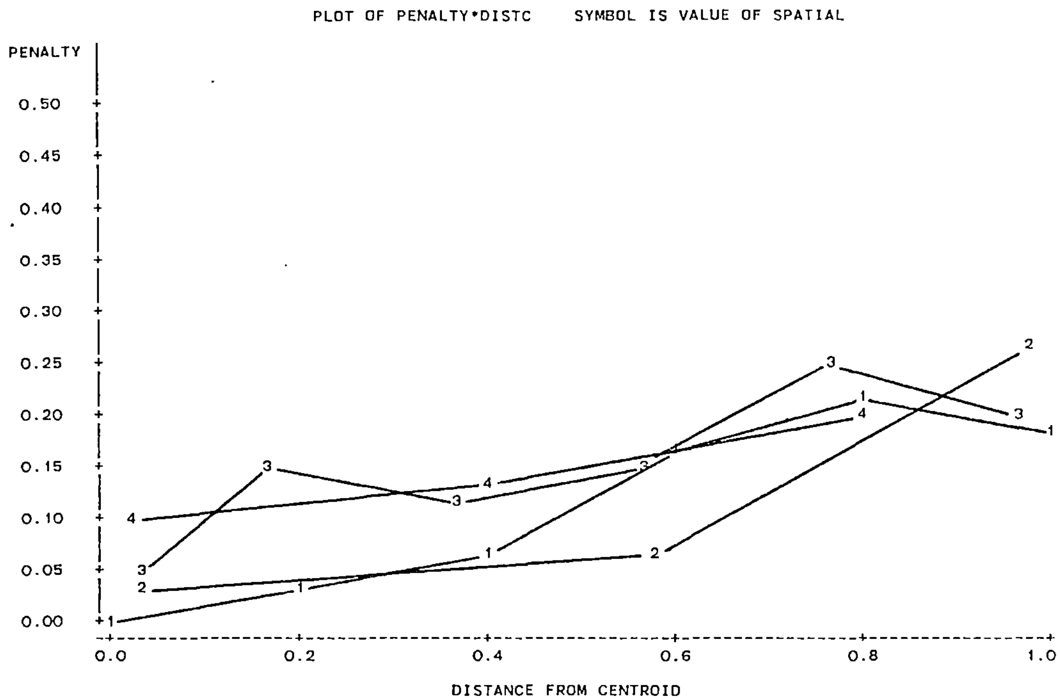


(C) THIRD DEMAND PATTERN

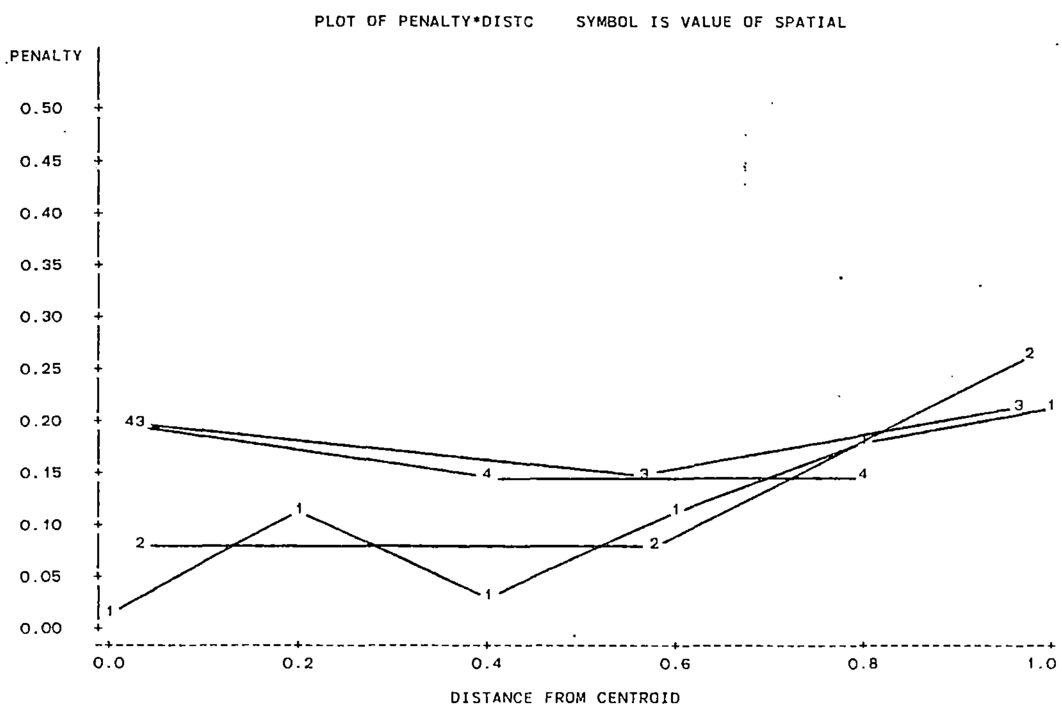


(D) FOURTH DEMAND PATTERN

Figure 6.6. (Continued)

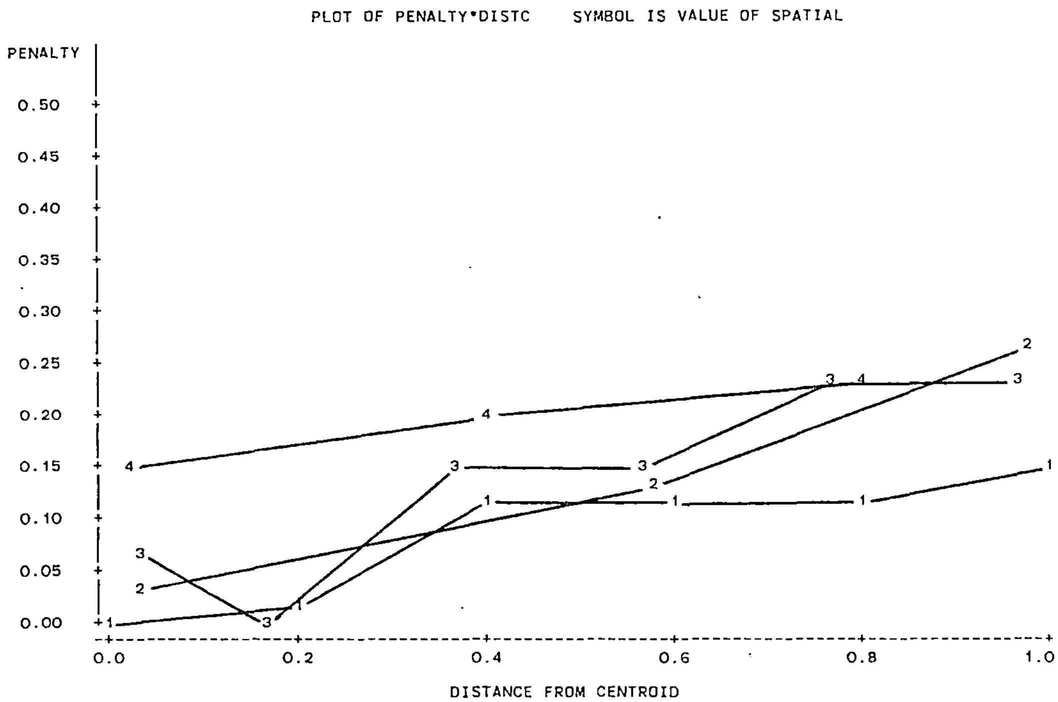


(A) FIRST DEMAND PATTERN

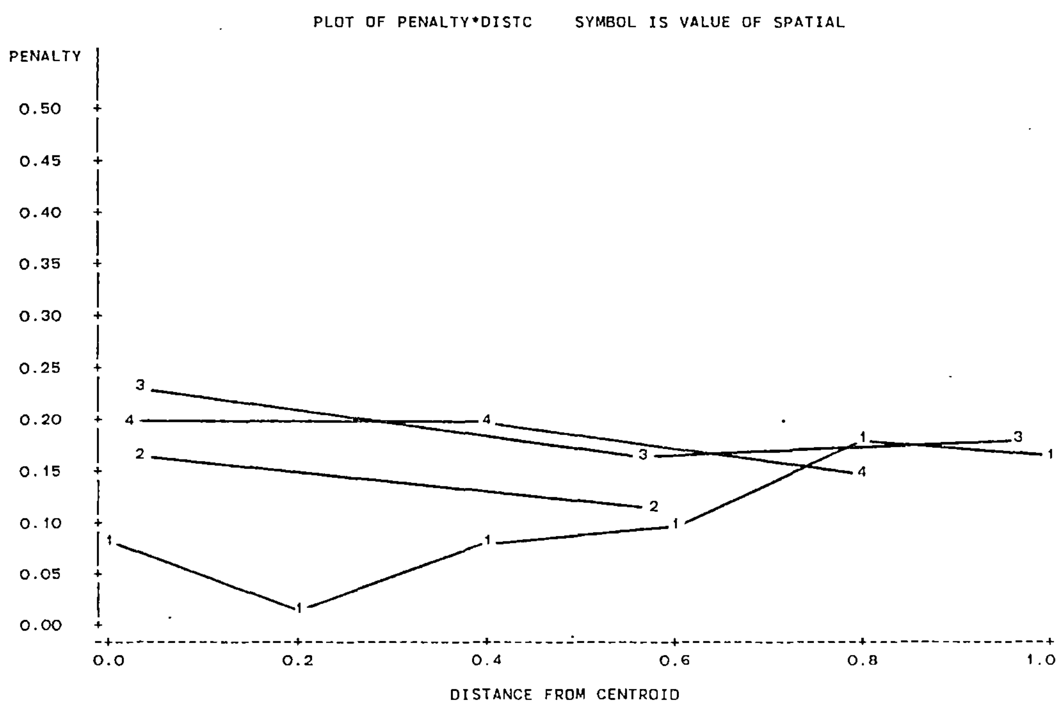


(B) SECOND DEMAND PATTERN

Figure 6.7. Route-Length Balancing Penalties

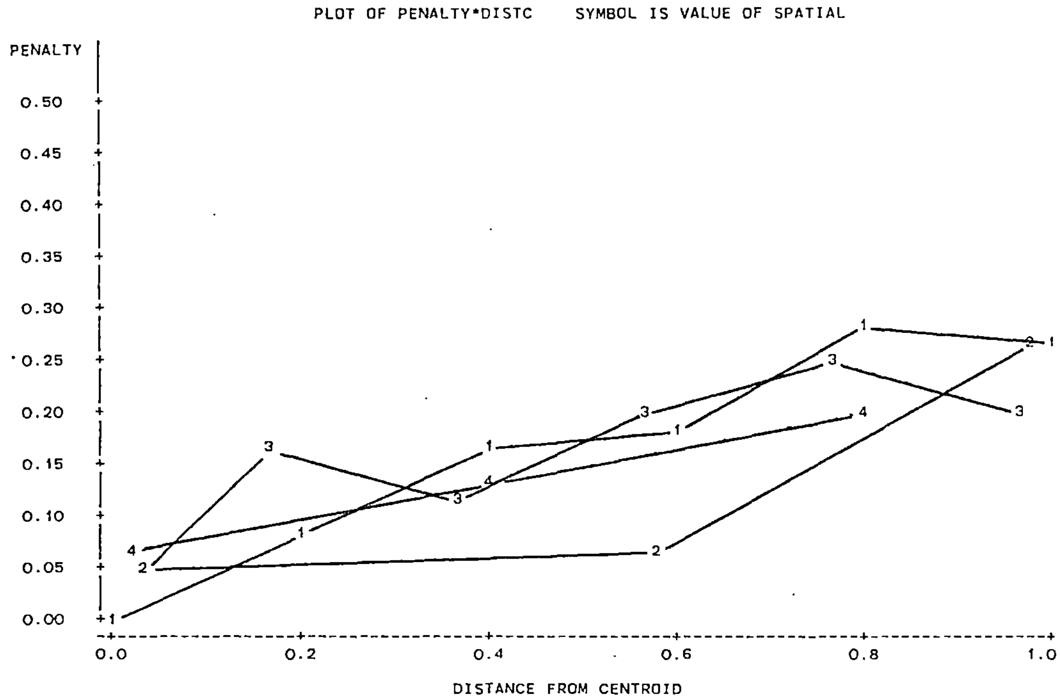


(C) THIRD DEMAND PATTERN

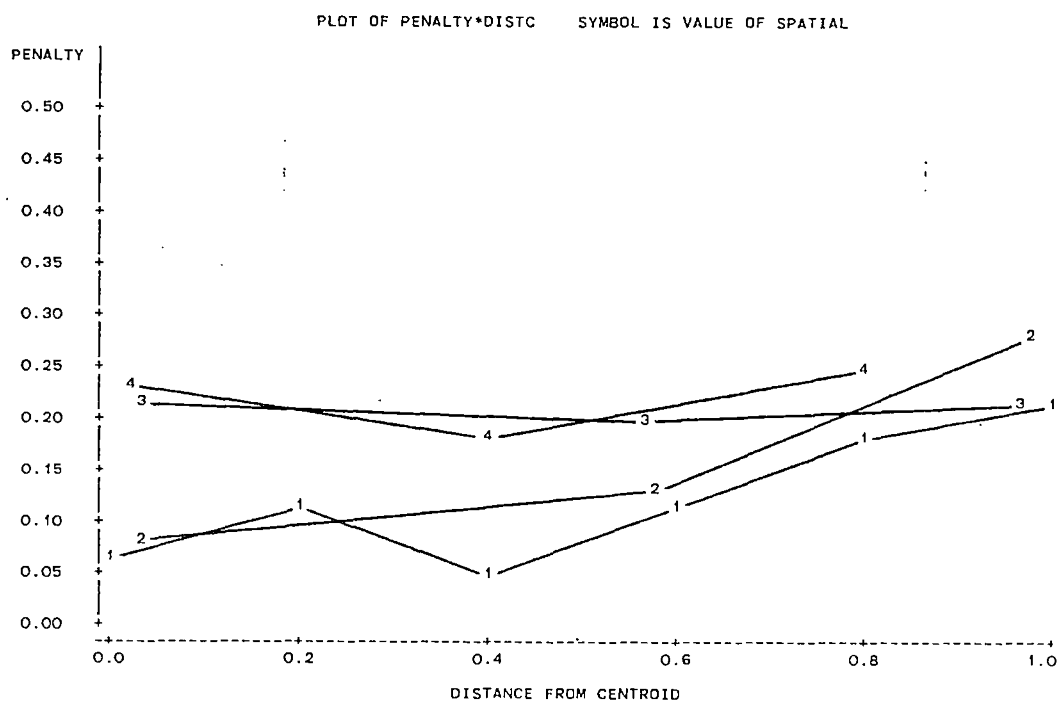


(D) FOURTH DEMAND PATTERN

Figure 6.7. (Continued)

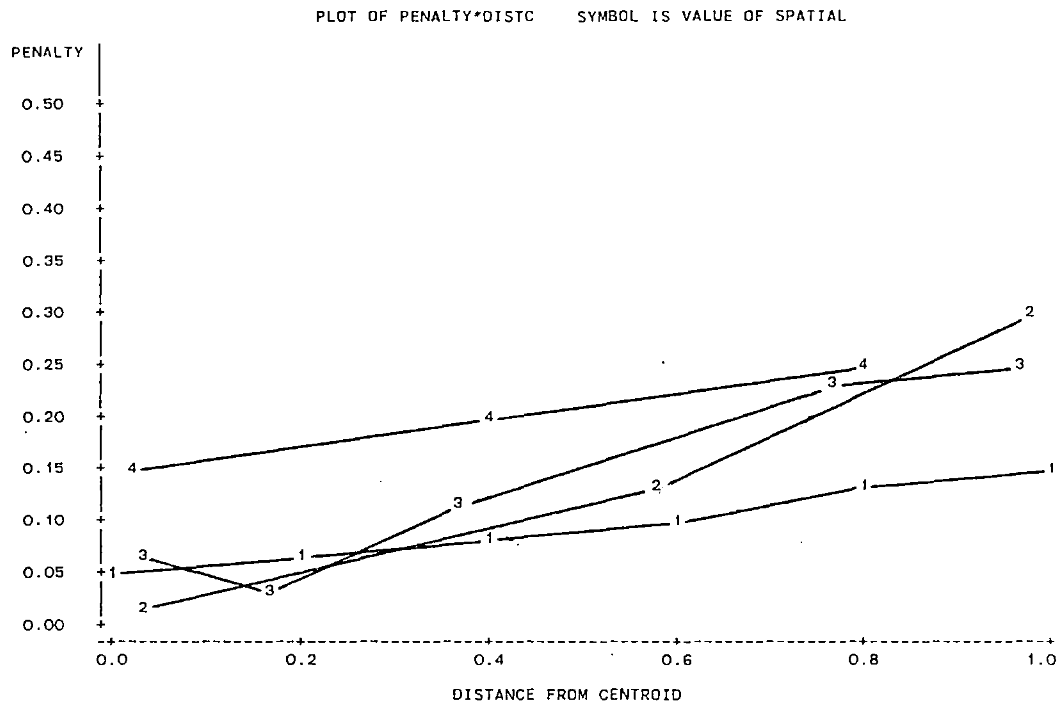


(A) FIRST DEMAND PATTERN

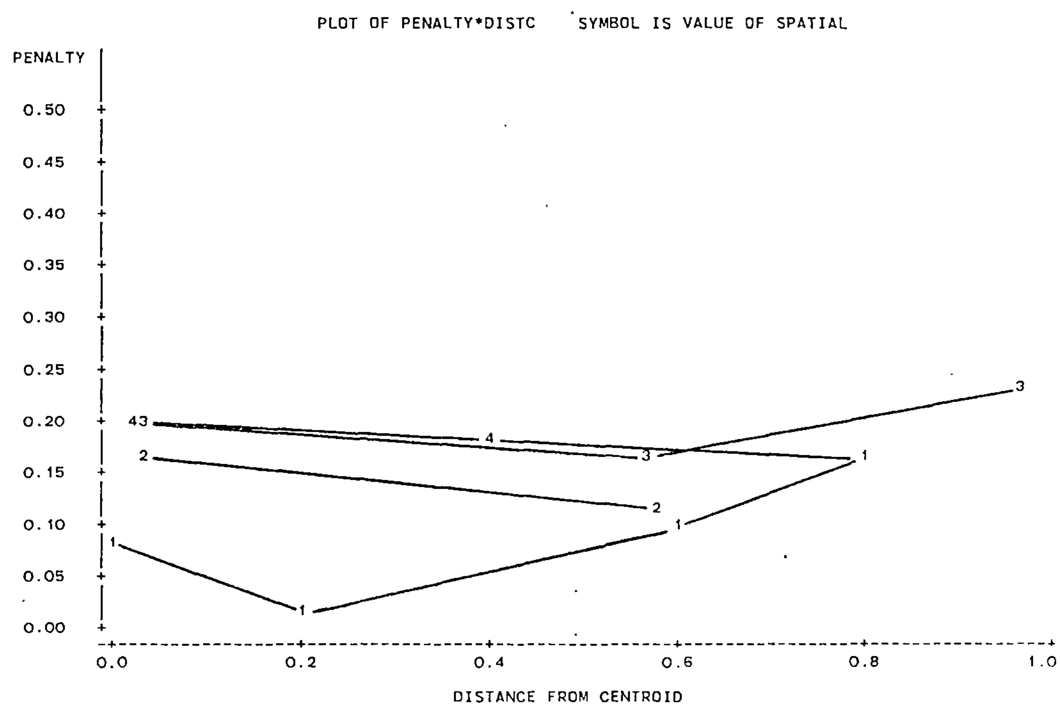


(B) SECOND DEMAND PATTERN

Figure 6.8. Penalties for Balancing Both Route-Load and Route-Length



(C) THIRD DEMAND PATTERN



(D) FOURTH DEMAND PATTERN

Figure 6.8. (Continued)

situations in which demand is actually uniform, but also for those situations in which route load is measured by the number of customers on a route (e.g., certain cases of the driver-sell environment).

3. Balancing of route lengths costs less if the spatial pattern of customer locations is relatively structured.
4. Distribution systems which currently balance route lengths can also balance route loads at little or no added routing cost if minor tradeoffs are made between the two objectives.

Further research, employing a large number of different problems, is required to test more completely the validity of these statements.

CHAPTER VII

SUMMARY, CONCLUSIONS, AND RECOMMENDATIONS

Summary

The purpose of this research was to examine the use of multi-criteria analysis in solving the workload balanced vehicle routing problem (WBVRP), a problem which was not found in the literature during an extensive search by the author. Four specific goals were established in Chapter I to accomplish this.

Goal Number One

The first goal was to develop a multiobjective model structure to solve the WBVRP, utilizing user interaction to make necessary tradeoffs among the three objectives of the problem. This model structure was developed in Chapter III. It is based on a heuristic implementation of the Method of Satisfactory Goals (Benson, 1975). Using this method, a problem is solved by minimizing a least satisfactory objective function at each stage, subject to satisfactory levels of the other objective functions being maintained.

Goal Number Two

The second goal was to develop and evaluate methods to minimize the three objective functions of the WBVRP. These algorithms were presented in Chapter IV. All three of these algorithms are based on arc exchange heuristics. For the distance minimization algorithm, the arc exchanges are used to provide reductions in the total distance objective; for the workload deviation minimization algorithms, the arc exchanges are used to cluster customers into potential routes. Representative solution times for these algorithms on an IBM 3081D are shown in Table 7.1. For the distance minimization algorithm, solution times are highly dependent upon the number of customers in the problem. For the route-length deviation algorithm, the solution times are highly dependent upon the total number of TSPs solved by the algorithm. The solution times for the route-load deviation algorithm are not as dependent upon the number of TSPs solved. In either case, only about one-half of the variability in solution times for the deviation minimization algorithms could be explained by regression models.

The solution quality of the three algorithms as a function of the number of runs is also shown in Table 7.1. For the distance minimization algorithm, the measure of solution quality is expressed as a percentage difference from the best known solution. For the deviation minimization algorithms, the solution quality is expressed as a percentage of the maximum possible improvement in deviation.

Goal Number Three

The third goal was to incorporate the multiobjective model structure into an interactive computer program, and to evaluate the performance of the program in terms of efficiency (solution times) and effectiveness (solution values). The interactive computer program was presented in Chapter V. This program uses goal tradeoffs to estimate the effects of constraint relaxation on the minimization of the least

TABLE 7.1

SOLUTION TIMES AND SOLUTION QUALITY OF SINGLE-OBJECTIVE ALGORITHMS

Algorithm	Minimum Problem Size	CPU Sec.	Maximum Problem Size	CPU Sec.	Solution Quality 1 Run	2 Runs	3 Runs
Total Distance	22 cities	0.15	100 cities	17.91	3.75	3.17	3.14
Route-Length Deviation	315 TSPs	0.94	13,101 TSPs	16.10	91.22	93.40	94.72
Route-Load Deviation	2 TSPs	0.28	3,648 TSPs	5.39	82.83	83.93	92.92

satisfactory objective function. A convergence analysis using utility functions showed the program to yield final solutions which were close to one another when the procedure was begun with different starting solutions, although no identical solutions were generated. A nondominance analysis showed about fifteen percent of the solutions generated by the procedure to be dominated. The majority of problems were solved in less than one minute CPU time.

Goal Number Four

The fourth goal was to determine the penalty which must be paid by distribution managements in order to balance route lengths and route loads under differing patterns of demand and customer location. In Chapter VI, a standard 36-city problem was set up and solved using different depot locations, customer demand patterns, and customer location patterns. Penalties were calculated as a fraction of total routing costs without workload balancing.

Conclusions

The heuristic version of the Method of Satisfactory Goals as presented herein appears to offer good satisficing solutions to the WBVRP. The procedure is straightforward and easy to use, and the amount of user input at each step is minimal. Solution times are not excessive for the quality of solutions obtained. To improve the solution quality, the following are required:

1. Decrease the error of the single-objective algorithms by additional runs of the appropriate algorithm in each iteration.

2. Provide better estimates of the effect of constraint relaxation in the two constraining achievement levels.

However, these improvements could lead to substantially increased computing times.

From the analysis of workload balancing costs, only conclusions regarding the standard 36-city problem of Chapter VI can be made. More analyses must be performed before conclusions can be extended to WBVRPs in general. However, the analysis of the 36-city problem points to the likelihood that distribution systems with centrally located depots are candidates for workload balancing, particularly if the demand is constant. Moreover, it is likely that those distribution systems currently balancing route lengths can also balance route loads with little or no additional routing costs.

Recommendations for Further Research

Attempts to examine the effects of problem characteristics on the cost of workload balancing were only partially successful. Much more research is needed to identify the characteristics of a problem which make it a likely or unlikely candidate for workload balancing. In addition, specific cases need to be solved in a real world setting. These cases need to evaluate not only the effectiveness of the solution technique, but also the actual benefits derived from workload balancing.

Specific modifications to the procedure developed in this research can be suggested for evaluation. In particular, the following should be explored:

1. Use of an exact TSP procedure inside the deviation minimization routines instead of the 3-opt method employed herein. Because

of the time required by exact methods, an efficient way of filtering out unpromising exchanges could be employed to reduce the total number of TSPs to be solved.

2. Use of 4-arc exchanges instead of 3-arc exchanges in the deviation minimization clustering procedures to provide more types of customer exchanges between routes. To offset the greater number of arc combinations involved, the procedure could be modified to consider exchanges between only two routes at a time.

3. A different means of estimating the effect of relaxing one or both of the two constraining achievement levels at each iteration of the procedure.

Other multicriteria approaches to solving the WBVRP can be investigated. One such approach would be to formulate the problem as a multiple objective integer program, and then use heuristics to solve a single objective integer program in each iteration of a procedure such as described by Gabbaini and Magazine (1985).

A natural extension of the WBVRP is the workload balanced vehicle scheduling problem (WBVSP), in which temporal constraints are considered. The problem might involve time windows during which deliveries must be made, or might allow a longer period of time (e.g., a week) in which to balance the workload elements.

Deterministic demands have been assumed in all of the WBVRPs solved herein. Future research could involve stochastic versions of either the WBVRP or the WBVSP, to account for the variability of demand at each customer location.

Finally, research in the use of exact methods to solve this and other vehicle routing problems could be undertaken. Early researchers found the computing times to obtain optimal solutions to be prohibitive, and resorted to the development of innovative heuristics to arrive at "good" solutions, instead. This approach has continued over the years, although computer technology has achieved CPU speeds many thousands of times faster than was available to those earlier researchers. Over the same period, computational costs have declined by a factor of several thousands. Efforts should be made to exploit these gains in computer technology by solving at least some aspects of vehicle routing problems in an optimal manner.

BIBLIOGRAPHY

Balas, E. and N. Christofides. "A Restricted Lagrangean Approach to the Traveling Salesman Problem." Mathematical Programming, 21 (1981), 19-46.

Balinski, M. and R. Quandt. "On an Integer Program for a Delivery Problem." Operations Research, 12 (1964), 300-304.

Ball, M. O., B. L. Golden, A. A. Assad, L. D. Bodin. "Planning for Truck Fleet Size in the Presence of a Common-Carrier Option." Decision Sciences, 14 (1983), 103-120.

Beasley, J. E. "Route First - Cluster Second Methods for Vehicle Routing." OMEGA, 11 (1983), 403-408.

Bellman, R. "Dynamic Programming Treatment of the Travelling Salesman Problem." J. ACM, 9 (1962), 61-63.

Bellmore, M. and J. Malone. "Pathology of Traveling-Salesman Subtour-Elimination Algorithms." Operations Research, 13 (1971), 278-307.

Bellmore, M. and G. Nemhauser. "The Traveling Salesman Problem: A Survey." Operations Research, 16 (1974), 538-558.

Beltrami, E. and L. Bodin. "Networks and Vehicle Routing for Municipal Waste Collection." Networks, 4 (1974), 65-94.

Benayoun, R., J. de Montgolfier, J. Tergny, and O. Larichev. "Linear Programming with Multiple Objective Functions: Step Method (STEM)." Mathematical Programming, 1 (1971), 366-375.

Benson, R. G. "Interactive Multiple Criteria Optimization Using Satisfactory Goals." (Ph.D. dissertation, University of Iowa, 1975.)

Benton, W. C. "Evaluating a Modified Heuristic for the Multiple-Vehicle Scheduling Problem." IIE Transactions, 18 (1986), 70-78.

Bodin, L. and L. Berman. "Routing and Scheduling of School Buses by Computer." Transportation Science, 13 (1979), 113-125.

Bodin, L., B. Golden, A. Assad, M. Ball. "Routing and Scheduling of Vehicles and Crews." Computers and Operations Research, 10 (1983), 63-211.

Bodin, L. and B. Golden. "Classification in Vehicle Routing and Scheduling." Networks, 11 (1981), 97-108.

Boychuk, L. M. and V. O. Ovchinnikov. "Principal Methods for Solution of Multicriterial Optimization Problems (Survey)." Soviet Automatic Control, 6 (1973), 1-4.

Chapleau, L., J. Ferland, J. M. Rousseau. "Clustering for Routing in Dense Areas." University of Montreal Transportation Research Center Publication No. 234, (1981).

Charnes, A. and W. W. Cooper. Management Models and Industrial Applications of Linear Programming. New York: John Wiley, 1961.

Cheshire, I. M., A. M. Malleson, P. F. Naccache. "A Dual Heuristic for Vehicle Scheduling." J. Operational Research Society, 33 (1982), 51-61.

Christofides, N. "Bounds for the Travelling-Salesman Problem." Operations Research, 20 (1972), 1044-1056.

Christofides, N. "The Travelling Salesman Problem." Combinatorial Optimization. Ed. N. Christofides, R. Mingozzi, P. Toth, and C. Sandi. New York: Wiley, 1979, pp. 131-149.

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

Christofides, N. and S. Eilon. "Algorithms for Large-Scale Travelling Salesman Problems." Operational Research Quarterly, 23 (1972), 511-518.

Christofides, N., A. Mingozzi, P. Toth. "Exact Algorithms for the Vehicle Routing Problem, Based on Spanning Tree and Shortest Path Relaxations." Mathematical Programming, 20 (1981), 255-282.

Clarke, G. and J. W. Wright. "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points." Operations Research, 12 (1964), 568-581.

Croes, G. A. "A Method for Solving Traveling-Salesman Problems." Operations Research, 6 (1958), 791-812.

Dantzig, G., D. Fulkerson, S. Johnson. "Solution of a Large-Scale Traveling Salesman Problem." Operations Research, 2 (1954) 393-410.

Dantzig, G. and J. Ramser. "The Truck Dispatching Problem." Management Science, 6 (1959), 81-91.

Dileepan, P. "Delivery Planning Problem." (Ph.D. dissertation, University of Houston, 1984.)

Dyer, J. S. "Interactive Goal Programming." Management Science, 19 (1972), 62-70.

Eastman, W. L. "Linear Programming with Pattern Constraints." (Ph.D. dissertation, Harvard University, 1958).

Eilon, S., C. Watson-Gandy, N. Christofides. Distribution Management: Mathematical Modeling and Practical Analysis. New York: Hafner, 1971.

Evans, S. R. and J. P. Norback. "An Heuristic Method for Solving Time-Sensitive Routing Problems." J. Operational Research Society, 35 (1984), 407-414.

Farquhar, P. H. "A Survey of Multiattribute Utility Theory and Applications." Multiple Criteria Decision Making. Ed. M. K. Starr and M. Zeleny. New York: North Holland, 1977, pp. 59-90.

Fichefet, J. "GPSTEM: An Interactive Multiobjective Optimization Method." Progress in Operation Research, Vol. 1. Ed. A. Prekopa. Amsterdam: North Holland, 1976, pp. 317-332.

Fishburn, P. C. "Lexicographic Orders, Utilities, and Decision Rules: A Survey." Management Science, 20 (1974), 1442-1471.

Fisher, M. "The Lagrangian Relaxation Method for Solving Integer Programming Problems." Management Science, 27 (1981), 1-12.

Fisher, M. and R. Jaikumar. "A Generalized Assignment Heuristic for Vehicle Routing." Networks, 11 (1981), 109-124.

Flood, M. "The Traveling Salesman Problem." Operations Research, 4 (1956), 61-75.

Foster, B. and D. Ryan. "An Integer Programming Approach to the Vehicle Scheduling Problem." Operational Research Quarterly, 27 (1976), 367-384.

Gabbani, D. and M. Magazine. "An Interactive Heuristic Approach for Multi-Objective Integer Programming Problems." Department of Management Sciences Working Paper, University of Waterloo, 1985.

Gal, T.and J. Nedoma. "Multiparametric Linear Programming." Management Science, 18 (1972), 406-421.

Gaskell, T. "Bases for Vehicle Fleet Scheduling." Operational Research Quarterly, 18 (1967), 281-295.

Geoffrion, A. M., J. S. Dyer, and A. Feinberg. "An Interactive Approach for Multi-Criterion Optimization, with an Application to the Operation of an Academic Department." Management Science, 19 (1972), 357-368.

Gillett, B. and J. Johnson. "Multi-terminal Vehicle-dispatch Algorithm." OMEGA, 4 (1976), 711-718.

Gillett, B. and L. Miller. "A Heuristic Algorithm for the Vehicle Dispatch Problem." Operations Research, 22 (1974), 340-349.

Golden, B., A. Assad, L. Levy, F. Gheysens. "The Fleet Size and Mix Vehicle Routing Problem." Computers and Operations Research, 11 (1984), 49-66.

Golden, B., L. Bodin, T. Doyle, W. Stewart. "Approximate Traveling Salesman Algorithms." Operations Research, 28 (1980), 694-711.

Golden, B., T. Magnanti, H. Nguyen. "Implementing Vehicle Routing Algorithms." Networks, 7 (1977), 113-148.

Haimes, Y. Y., W. A. Hall, and H. T. Freedman. Multiobjective Optimization in Water Resources Systems, the Surrogate Worth Tradeoff Method. New York: Elsevier Scientific, 1975.

Hansen, K. and J. Krarup. "Improvements of the Held-Karp Algorithm for the Symmetric Traveling Salesman Problem." Mathematical Programming, 7 (1974), 87-96.

Held, M. and R. Karp. "A Dynamic Programming Approach to Sequencing Problems." J. SIAM, 10 (1962), 196-210.

Held, M. and R. Karp. "The Traveling Salesman Problem and Minimum Spanning Trees." Operations Research, 18 (1970), 1138-1162.

Held, M. and R. Karp. "The Traveling Salesman Problem and Minimum Spanning Trees - Part II." Mathematical Programming, 1 (1971), 6-25.

Holmes, R. and R. Parker. "A Vehicle Scheduling Procedure Based Upon Savings and a Solution Perturbation Scheme." Operational Research Quarterly, 27 (1976), 83-92.

Husban, Ahmad O. "Balancing Routes in a Class of Vehicle Routing Problems." (Ph.D. dissertation, Rensselaer Polytechnic Institute, 1985.)

Hwang, Ching-Lai and Abu S. Masud. Multiple Objective Decision Making - Methods and Applications. Berlin: Springer-Verlag, 1979.

Ignizio, J. P. Goal Programming and Extensions. Massachusetts: Lexington Books, 1976.

Karg, L. and G. Thompson. "A Heuristic Approach to Solving Traveling Salesman Problems." Management Science, 10 (1964), 225-248.

Keeny, R. L. "Utility Functions for Multiattributed Consequences." Management Science, 18 (1972), 276-287.

Kirby, R. and J. McDonald. "The Savings Method for Vehicle Scheduling." Operational Research Quarterly, 24 (1973), 305.

Klein, D. and E. Hannan. "An Algorithm for the Multiple Objective Integer Linear Programming Problem." European J. of Operational Research, 9 (1982), 378-385.

Krolak, P., W. Felts, J. Nelson. "A Man-Machine Approach Toward Solving the Generalized Truck Dispatching Problem." Transportation Science, 6 (1972), 149-170.

Lawrence, J. L. "Interactive Vehicle Dispatching: A Hybrid Approach." (Ph.D. dissertation, University of Missouri - Rolla, 1981.)

Lee, S. "Interactive Integer Goal Programming: Methods and Application." Multiple Criteria Problem Solving. Ed. S. Zionts. New York: Springer-Verlag, 1978, pp. 362-383.

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

Lin, S. and B. Kernighan. "An Effective Heuristic Algorithm for the Traveling Salesman Problem." Operations Research, 21 (1973), 498-516.

Little, J., K. Murty, D. Sweeny, C. Karel. "An Algorithm for the Traveling Salesman Problem." Operations Research, 11 (1963), 972-989.

Martin, G. T. "An Accelerated Euclidean Algorithm for Integer Linear Programming." Recent Advances in Mathematical Programming. Eds. R. L. Graves and P. Wolfe. 1963, pp. 311-318.

Miliotis, P. "Integer Programming Approaches to the Traveling Salesman Problem." Mathematical Programming, 10 (1976), 367-378.

Miller, C., A. Tucker, R. Zemlin. "Integer Programming Formulation of Travelling Salesman Problems." J. ACM, 7 (1960), 326-332.

Mole, R. "A Survey of Local Delivery Vehicle Routing Methodology." J. Operational Research Society, 30 (1979), 245-252.

Mole, R. and S. Jameson. "A Sequential Route-Building Algorithm Employing a Generalized Savings Criterion." Operational Research Quarterly, 27 (1976), 503-511.

Newton, R. and W. Thomas. "Bus Routing in a Multi-School System." Computers and Operations Research, 1 (1974), 213-222.

Norback, J. and R. Love. "Geometric Approaches to Solving the Traveling Salesman Problem." Management Science, 23 (1977), 1208-1223.

Park, Y. "Solutions to Vehicle Routing Problems in a Multiple Criteria Environment." (Ph.D. dissertation, Oklahoma State University, Stillwater, 1984).

Pierce, J. "Direct Search Algorithms for Truck-Dispatching Problems Part I." Transportation Research, 3 (1969), 1-42.

Reeves, G. R. and L. S. Franz. "A Simplified Interactive Multiple Objective Linear Programming Procedure." Computers and Operations Research, 12 (1985), 589-601.

Reiter, S. and G. Sherman. "Discrete Optimizing." J. SIAM, 13 (1965), 864-889.

Rosenkrantz, D., R. Stearns, P. Lewis. "Approximate Algorithms for the Traveling Salesperson Problem." Proc. of 15th Annual IEEE Symposium on Switching and Automata Theory, (1974), pp. 33-42.

Russell, R. "An Effective Heuristic for the M-tour Traveling Salesman Problem with Some Side Conditions." Operations Research, 25 (1977), 517-524.

Shapiro, D. "Algorithms for the Solution of the Optimal Cost Travelling Salesman Problem." (Ph.D. dissertation, Washington University, St. Louis, 1966.)

Shapiro, Jeremy F. Mathematical Programming: Structures and Algorithms. New York: Wiley, 1979.

Stacey, P. "Practical Vehicle Routing Using Computer Programs." J. Operational Research Society, 34 (1983), 975-981.

Steur, R. "An Interactive Multiple Objective Linear Programming Procedure." TIMS Studies in the Management Sciences, 6 (1977), 225-239.

Stewart, W. "New Algorithms for Deterministic and Stochastic Vehicle Routing Problems." (D.B.A. dissertation, University of Maryland, 1981.)

Stewart, W. "An Accelerated Branch Exchange Heuristic for the Traveling Salesman Problem." School of Business Administration Working Paper No. 85-004, College of William and Mary, Williamsburg, 1985.

Stewart, W. and B. Golden. "A Lagrangean Relaxation Heuristic for Vehicle Routing." European J. of Operational Research, 15 (1984), 84-88.

Tillman, F. and T. Cain. "An Upper Bounding Algorithm for the Single and Multiple Terminal Delivery Problem." Management Science, 18 (1972), 664-682.

Tillman, F. and H. Cochran. "A Heuristic Approach for Solving the Delivery Problem." J. of Industrial Engineering, 19 (1968), 354-358.

Turner, W., P. Ghare, L. Foulds. "Transportation Routing Problem - A Survey." AIIE Transactions, 6 (1974), 288-301.

Tyagi, M. "A Practical Method for the Truck Dispatching Problem." J. Operations Research Society of Japan, 10 (1968), 76-92.

Waters, C. G. J. "Interactive Vehicle Routing." J. Operational Research Society, 35 (1984), 821-826.

Webb, M. "Some Methods of Producing Approximate Solutions to Traveling Salesman Problems with Hundreds or Thousands of Cities." Operational Research Quarterly, 22 (1971), 49-66.

Williams, B. W. "Vehicle Scheduling: Proximity Priority Scheduling." J. Operational Research Society, 33 (1982), 961-966.

Wiorkowski, J. and K. McElvain. "A Rapid Heuristic Algorithm for the Approximate Solution of the Traveling Salesman Problem." Transportation Research, 9 (1975), 181-185.

Wren, A. and A. Holliday. "Computer Scheduling of Vehicles from One or More Depots to a Number of Delivery Points." Operational Research Quarterly, 23 (1972), 333-344.

Yellow, P. "A Computational Modification to the Savings Method of Vehicle Scheduling." Operational Research Quarterly, 21 (1970), 281-283.

Yu, P. L. and M. Zeleny. "The Set of all Non-Dominated Solutions in Linear Cases and a Multicriteria Simplex Method." Journal of Mathematical Analysis and Applications, 49 (1975), 430-468.

Zeleny, M. Multiple Criteria Decision Making. New York: McGraw-Hill, 1982.

Zionts, S. "Integer Linear Programming with Multiple Objectives." Annals of Discrete Mathematics, 1 (1977), 551-562.

Zionts, S. and J. Wallenius. "An Interactive Programming Method for Solving the Multiple Criteria Problem." Management Science, 22 (1976), 652-663.

APPENDIXES

APPENDIX A

APPROXIMATE NUMBER OF QUALIFIED ARC COMBINATIONS IN DEVIATION MINIMIZATION ALGORITHMS

APPROXIMATE NUMBER OF QUALIFIED ARC COMBINATIONS IN DEVIATION MINIMIZATION ALGORITHMS

There are $\binom{N+R}{3}$ total arc combinations in a 3-arc exchange algorithm, where N is the number of customers in the problem and R is the number of routes. To minimize the maximum deviation between routes, only certain of these combinations are 'qualified' for consideration. To be qualified, a combination (1) must have at least one arc in the largest route or in the smallest route, and (2) cannot have all three arcs in the same route. To find an approximate number of qualified arc combinations, all routes are assumed to be of about the same size so that the probability of a single arc being in a given route is $\frac{1}{R}$. The easiest way of finding the number of qualified combinations is to first find the number of unqualified combinations and subtract that number from the total number of combinations. A combination is unqualified for consideration if any of the following apply: (1) all three arcs are in the smallest route, (2) all three arcs are in the largest route, or (3) all three arcs are contained elsewhere.

First, the total number of combinations is found.

$$\text{Combinations (total)} = \binom{N+R}{3} \tag{A.1}$$

or

$$\text{Combinations (total)} = \frac{(N+R)(N+R-1)(N+R-2)}{6} \tag{A.2}$$

which, if N+R is sufficiently large, can be approximated by

$$\text{Combinations (total)} = \frac{(N+R)^3}{6}\ . \tag{A.3}$$

Next, the total number of combinations of three arcs in the large route is found.

$$\text{Combinations (3 in large route)} = \binom{\frac{N}{R}+1}{3} \tag{A.4}$$

or $$\text{Combinations (3 in large route)} = (\frac{N}{R}+1)(\frac{N}{R})\ (\frac{N}{R}-1) \tag{A.5}$$

or $$\text{Combinations (3 in large route)} = \frac{N^3 - NR^2}{6R^3}\ . \tag{A.6}$$

Since it is assumed that routes are of approximately equal size, the number of combinations of three arcs in the small route is also given by (A.6). Therefore

$$\text{Combinations (3 in large or small route)} = \frac{2(N^3 - NR^2)}{6R^3}. \tag{A.7}$$

Now the total number of combinations of three arcs contained outside of the large or small routes is given by

$$\text{Combinations (3 elsewhere)} = \binom{N+R-2(\frac{N}{R}+1)}{3} \tag{A.8}$$

or $$\text{Combinations (3 elsewhere)} = \binom{N+R-\frac{2N}{R}-2}{3} \tag{A.9}$$

or $$\text{Combinations (3 elsewhere)} = \frac{(N+R-\frac{2N}{R}-2)(N+R-\frac{2N}{R}-3)(N+R-\frac{2N}{R}-4)}{6} \tag{A.10}$$

Once again, assuming N+R sufficiently large, the equation above can be approximated by

$$\text{Combinations (3 elsewhere)} = \frac{(NR+R^2-2N)^3}{6R^3}\ . \tag{A.11}$$

Now, the number of qualified combinations is given by

Q = Combinations (total) - Combinations (3 in large or small route)

- Combinations (3 elsewhere)

or $$Q = \frac{(N+R)^3}{6} - \frac{2(N^3-NR^2)}{6R^3} - \frac{(NR+R^2-2N)^3}{6R^3}\ . \tag{A.12}$$

APPENDIX B

VEHICLE ROUTING PROBLEM DATA

TABLE B.1

GASKELL'S 22-CITY PROBLEM

City	X	Y	Demand	City	X	Y	Demand
1	295	272	125	12	267	242	300
2	301	258	84	13	259	265	250
3	309	260	60	14	315	233	500
4	217	274	500	15	329	252	150
5	218	278	300	16	318	252	100
6	282	267	175	17	329	224	250
7	242	249	350	18	267	213	120
8	230	262	150	19	275	192	600
9	249	268	1100	20	303	201	500
10	256	267	4100	21	208	217	175
11	265	257	225	22	326	181	75

Vehicle Capacity = 4500

Maximum Miles = 240

Allowance = 10 miles

Depot Coordinates: 266, 235

TABLE B.2

GASKELL'S 29-CITY PROBLEM

City	X	Y	Demand	City	X	Y	Demand
1	218	382	300	16	119	357	150
2	218	358	3100	17	115	341	100
3	201	370	125	18	153	351	150
4	214	371	100	19	175	363	400
5	224	370	200	20	180	360	300
6	210	382	150	21	159	331	1500
7	104	354	150	22	188	357	100
8	126	338	450	23	152	349	300
9	119	340	300	24	215	389	500
10	129	349	100	25	212	394	800
11	126	347	950	26	188	393	300
12	125	346	125	27	207	406	100
13	116	355	150	28	184	410	150
14	126	335	150	29	207	392	1000
15	125	355	550				

Vehicle Capacity = 4500

Maximum Miles = 240

Allowance = 10 Miles

Depot Coordinates: 162, 354

TABLE B.3

GASKELL'S 32-CITY PROBLEM

City	X	Y	Demand	City	X	Y	Demand
1	298	427	700	17	297	410	550
2	309	445	400	18	315	407	650
3	307	464	400	19	314	406	200
4	336	475	1200	20	321	391	400
5	320	439	40	21	321	398	300
6	321	437	80	22	314	394	1300
7	322	437	2000	23	313	378	700
8	323	433	900	24	304	382	750
9	324	433	600	25	295	402	1400
10	323	429	750	26	283	406	4000
11	314	435	1500	27	279	399	600
12	311	442	150	28	271	401	1000
13	304	427	250	29	264	414	500
14	293	421	1600	30	277	439	2500
15	296	418	450	31	290	434	1700
16	261	384	700	32	319	433	1100

Vehicle Capacity = 8000

Maximum Miles = 240

Allowance = 10 Miles

Depot Coordinates: 292, 425

TABLE B.4

CHRISTOFIDES AND EILON'S 50-CITY PROBLEM

City	X	Y	Demand	City	X	Y	Demand
1	37	52	7	26	27	68	7
2	49	49	30	27	30	48	15
3	52	64	16	28	43	67	14
4	20	26	9	29	58	48	6
5	40	30	21	30	58	27	19
6	21	47	15	31	37	69	11
7	17	63	19	32	38	46	12
8	31	62	23	33	46	10	23
9	52	33	11	34	61	33	26
10	51	21	5	35	62	63	17
11	42	41	19	36	63	69	6
12	31	32	29	37	32	22	9
13	5	25	23	38	45	35	15
14	12	42	21	39	59	15	14
15	36	16	10	40	5	6	7
16	52	41	15	41	10	17	27
17	27	23	3	42	21	10	13
18	17	33	41	43	5	64	11
19	13	13	9	44	30	15	16
20	57	58	28	45	39	10	10
21	62	42	8	46	32	39	5
22	42	57	8	47	25	32	25
23	16	57	16	48	25	55	17
24	8	52	10	49	48	28	18
25	7	38	28	50	56	37	10

Vehicle Capacity = 160

Maximum Miles: None

Depot Coordinates: 30, 40

TABLE B.5

CHRISTOFIDES AND EILON'S 75-CITY PROBLEM

No.	x	y	q	No.	x	y	q	No.	x	y	q	No.	x	y	q
1	22	22	18	20	66	14	22	39	30	60	16	58	40	60	21
2	36	26	26	21	44	13	28	40	30	50	33	59	70	64	24
3	21	45	11	22	26	13	12	41	12	17	15	60	64	4	13
4	45	35	30	23	11	28	6	42	15	14	11	61	36	6	15
5	55	20	21	24	7	43	27	43	16	19	18	62	30	20	18
6	33	34	19	25	17	64	14	44	21	48	17	63	20	30	11
7	50	50	15	26	41	46	18	45	50	30	21	64	15	5	28
8	55	45	16	27	55	34	17	46	51	42	27	65	50	70	9
9	26	59	29	28	35	16	29	47	50	15	19	66	57	72	37
10	40	66	26	29	52	26	13	48	48	21	20	67	45	42	30
11	55	65	37	30	43	26	22	49	12	38	5	68	38	33	10
12	35	51	16	31	31	76	25	50	15	56	22	69	50	4	8
13	62	35	12	32	22	53	28	51	29	39	12	70	66	8	11
14	62	57	31	33	26	29	27	52	54	38	19	71	59	5	3
15	62	24	8	34	50	40	19	53	55	57	22	72	35	60	1
16	21	36	19	35	55	50	10	54	67	41	16	73	27	24	6
17	33	44	20	36	54	10	12	55	10	70	7	74	40	20	10
18	9	56	13	37	60	15	14	56	6	25	26	75	40	37	20
19	62	48	15	38	47	66	24	57	65	27	14				

q = demand in cwt

Vehicle Capacity = 7 tons

Maximum miles: None

Depot Coordinates: 40, 40

TABLE B.6

CHRISTOFIDES AND EILON'S 100-CITY PROBLEM

No.	x	y	q	No.	x	y	q	No.	x	y	q	No.	x	y	q
1	41	49	10	26	45	30	17	51	49	58	10	76	49	42	13
2	35	17	7	27	35	40	16	52	27	43	9	77	53	43	14
3	55	45	13	28	41	37	16	53	37	31	14	78	61	52	3
4	55	20	19	29	64	42	9	54	57	29	18	79	57	48	23
5	15	30	26	30	40	60	21	55	63	23	2	80	56	37	6
6	25	30	3	31	31	52	27	56	53	12	6	81	55	54	26
7	20	50	5	32	35	69	23	57	32	12	7	82	15	47	16
8	10	43	9	33	53	52	11	58	36	26	18	83	14	37	11
9	55	60	16	34	65	55	14	59	21	24	28	84	11	31	7
10	30	60	16	35	63	65	8	60	17	34	3	85	16	22	41
11	20	65	12	36	2	60	5	61	12	24	13	86	4	18	35
12	50	35	19	37	20	20	8	62	24	58	19	87	28	18	26
13	30	25	23	38	5	5	16	63	27	69	10	88	26	52	9
14	15	10	20	39	60	12	31	64	15	77	9	89	26	35	15
15	30	5	8	40	40	25	9	65	62	77	20	90	31	67	3
16	10	20	19	41	42	7	5	66	49	73	25	91	15	19	1
17	5	30	2	42	24	12	5	67	67	5	25	92	22	22	2
18	20	40	12	43	23	3	7	68	56	39	36	93	18	24	22
19	15	60	17	44	11	14	18	69	37	47	6	94	26	27	27
20	45	65	9	45	6	38	16	70	37	56	5	95	25	24	20
21	45	20	11	46	2	48	1	71	57	68	15	96	22	27	11
22	45	10	18	47	8	56	27	72	47	16	25	97	25	21	12
23	55	5	29	48	13	52	36	73	44	17	9	98	19	21	10
24	65	35	3	49	6	68	30	74	46	13	8	99	20	26	9
25	65	20	6	50	47	47	13	75	49	11	18	100	18	18	17

q = demand in cwt.

Vehicle Capacity = 10 tons

Maximum Miles: None

Depot Coordinates: 35, 35

TABLE B.7

EXAMPLE 33-CITY PROBLEM

City	X	Y	Demand	City	X	Y	Demand
1	3	3	15	18	24	14	10
2	8	5	31	19	25	18	15
3	16	4	10	20	15	18	27
4	6	8	9	21	4	20	35
5	12	7	40	22	11	20	15
6	24	7	35	23	20	20	22
7	9	10	14	24	3	22	6
8	14	10	10	25	8	22	25
9	20	10	3	26	13	22	41
10	21	11	17	27	15	23	30
11	3	13	15	28	18	22	43
12	5	14	26	29	5	24	10
13	5	15	20	30	8	26	23
14	12	14	15	31	15	27	39
15	10	16	11	32	20	25	10
16	12	17	5	33	25	25	14
17	18	16	17				

Vehicle Capacity = 150

Maximum Miles = 50

Depot Coordinates: 13, 16

APPENDIX C

DATA FOR REGRESSION MODELS

TABLE C.1

DATA FOR 3-OPT SOLUTION TIMES
REGRESSION MODEL

No. of Cities	CPU Seconds
22	0.115
22	0.203
22	0.237
22	0.169
22	0.191
22	0.138
29	0.316
29	0.488
29	0.491
29	0.381
29	0.456
29	0.267
32	0.339
32	0.340
32	0.313
32	0.323
32	0.526
32	0.345
50	2.980
50	1.482
50	3.031
50	2.801
50	2.832
50	3.444
75	5.430
75	5.930
75	5.300
75	6.304
75	5.250
75	5.400
100	16.070
100	29.990
100	15.770
100	28.440
100	21.270
100	16.090

TABLE C.2

DATA FOR ROUTE-LENGTH DEVIATION REGRESSION MODELS

N	R	B_{Ln}	B_{Ld}	RLX_{dist}	RLX_{Ld}	TSPs	CPU
22	5	.42	.96	.03	.001	7949	4.05
22	5	.42	.96	.03	.27	8220	4.19
22	5	.42	.96	.03	.64	7530	4.18
22	5	.42	.96	.03	.99	7533	4.16
22	5	.21	.82	.001	.001	3540	1.78
22	5	.21	.82	.001	.60	7047	3.46
22	5	.21	.82	.15	.41	11373	5.45
22	5	.21	.82	.15	.001	13737	6.73
29	4	.54	.29	.04	.001	5507	7.56
29	4	.54	.29	.04	.30	7060	9.77
29	4	.54	.29	.04	.77	7750	10.79
29	4	.54	.29	.04	.99	7775	10.86
29	4	.06	.36	.001	.001	2399	2.90
29	4	.06	.36	.001	.69	7235	10.01
29	4	.06	.36	.15	.30	5402	6.71
29	4	.06	.36	.15	.001	2758	4.14
32	4	.22	.22	.04	.001	2608	3.88
32	4	.22	.22	.04	.22	3226	4.88
32	4	.22	.22	.04	.53	5149	6.12
32	4	.22	.22	.04	.99	7724	7.74
32	4	.09	.09	.001	.001	539	.98
32	4	.09	.09	.001	.53	1057	1.72
32	4	.09	.09	.10	.10	1542	2.45
32	4	.09	.09	.10	.001	868	1.66
50	5	.45	.05	.001	.001	883	2.42
50	5	.45	.05	.001	.28	2322	6.05
50	5	.45	.05	.001	.61	2322	6.06
50	5	.45	.05	.001	.99	2322	6.07
50	5	.09	.03	.001	.001	315	.94
50	5	.09	.03	.001	.30	1338	3.26
50	5	.09	.03	.20	.10	1338	3.26
50	5	.09	.03	.20	.001	315	.94
75	10	.72	.06	.05	.001	3750	5.06
75	10	.72	.06	.05	.31	3357	3.82
75	10	.72	.06	.05	.69	3360	3.85
75	10	.72	.06	.05	.99	3357	3.50
75	10	.63	.06	.001	.001	2151	2.59
75	10	.23	.05	.001	.001	3227	4.31
75	10	.23	.05	.001	.50	4737	5.92
75	10	.23	.05	.20	.10	13101	16.10
75	10	.23	.05	.20	.001	6628	8.81

TABLE C.3

DATA FOR ROUTE-LOAD DEVIATION REGRESSION MODELS

N	R	B_{Ln}	B_{Ld}	RLX_{dist}	RLX_{Ln}	TSPs	CPU
22	5	.42	.96	.001	.001	3156	1.81
22	5	.42	.96	.15	.001	1265	.99
22	5	.37	.62	.001	.58	257	.22
22	5	.37	.62	.08	.99	362	.47
22	5	.21	.82	.001	.001	2041	1.03
22	5	.03	.79	.05	.18	1293	.98
22	5	.02	.78	.03	.19	2011	1.44
29	4	.29	.54	.15	.001	1377	1.79
29	4	.29	.54	.001	.001	3648	5.39
29	4	.06	.37	.01	.32	1787	2.51
29	4	.06	.36	.01	.33	1667	2.35
29	4	.06	.36	.001	.001	2635	3.52
29	4	.03	.48	.13	.03	3107	4.77
29	4	.03	.35	.13	.02	2418	4.14
29	4	.06	.23	.001	.06	1231	1.55
29	4	.06	.23	.08	.12	1231	1.55
32	4	.22	.20	.001	.001	1140	1.47
32	4	.09	.11	.01	.21	93	.37
32	4	.05	.15	.001	.24	28	.16
32	4	.09	.11	.001	.001	539	.96
32	4	.09	.11	.001	.001	490	.96
32	4	.03	.10	.01	.05	314	.75
32	4	.04	.09	.05	.04	75	.29
32	4	.22	.20	.15	.001	479	.95
32	4	.06	.10	.001	.06	451	.82
32	4	.06	.10	.08	.13	451	.82
50	5	.09	.03	.001	.99	5	.31
50	5	.07	.05	.01	.99	66	.61
50	5	.09	.03	.001	.001	94	.48
50	5	.08	.06	.001	.01	182	.70
50	5	.08	.06	.20	.01	107	.51
50	5	.09	.03	.20	.001	94	.48
50	5	.45	.05	.001	.001	278	1.02
50	5	.45	.05	.15	.001	23	.37
50	5	.45	.01	.001	.80	7	.29
50	5	.45	.01	.08	.99	2	.28
75	10	.37	.06	.001	.92	230	1.29
75	10	.63	.06	.001	.98	178	1.12
75	10	.63	.06	.001	.001	465	1.10
75	10	.23	.05	.001	.001	306	1.12
75	10	.23	.05	.001	.001	306	1.53
75	10	.10	.03	.09	.14	2	1.05

TABLE C.3 (Continued)

N	R	B_{Ln}	B_{Ld}	RLX_{dist}	RLX_{Ln}	TSPs	CPU
75	10	.23	.05	.20	.001	30	1.29
75	10	.72	.06	.001	.001	26	.97
75	10	.72	.04	.15	.001	19	1.30
75	10	.67	.02	.001	.99	1	1.04
75	10	.67	.02	.08	.99	1	1.04

APPENDIX D

FORTRAN PROGRAM LISTING

```
C
C
C
C**********************************************************************
C
C
C
C              FORTRAN PROGRAM TO SOLVE THE WORKLOAD
C                BALANCED VEHICLE ROUTING PROBLEM
C               (WBVRP), USING A HEURISTIC VERSION
C               OF THE METHOD OF SATISFACTORY GOALS.
C
C
C**********************************************************************
C
C
C
C                 AUTHOR:     J. D. ALLISON
C                 ADVISOR:    M. P. TERRELL
C                 COMPUTER:   IBM 3081D
C                 DATE:       JUNE, 1986
C
C
C       SCHOOL OF INDUSTRIAL ENGINEERING AND MANAGEMENT
C                OKLAHOMA STATE UNIVERSITY
C                STILLWATER, OKLAHOMA 74078
C
C
C**********************************************************************
C
C
C
C                  SUBROUTINES USED IN PROGRAM:
C
C     SAVNGS - CLARKE AND WRIGHT SAVINGS ALGORITHM.
C     TWOOPT - 2-OPT DISTANCE MINIMIZATION ALGORITHM.
C     THROPT - 3-OPT DISTANCE MINIMIZATION ALGORITHM.
C     LDDV2  - 2-ARC ROUTE LOAD DEVIATION ALGORITHM.
C     LDDV3  - 3-ARC ROUTE LOAD DEVIATION ALGORITHM.
C     LNDV2  - 2-ARC ROUTE LENGTH DEVIATION ALGORITHM.
C     LNDV3  - 3-ARC ROUTE LENGTH DEVIATION ALGORITHM.
C     FEAS2  - DETERMINES FEASIBILITY OF 2-ARC EXCHANGE.
C     XCHNG2 - PERFORMS 2-ARC EXCHANGE.
C     XCHNG3 - PERFORMS 3-ARC EXCHANGE.
C     FXCH2  - PERFORMS TEMPORARY 2-ARC EXCHANGE.
C     FXCH3  - PERFORMS TEMPORARY 3-ARC EXCHANGE.
C     TSP    - 3-OPT VERSION OF TRAVELING SALESMAN PROBLEM ALGORITHM.
C     NONDOM - ELIMINATES DOMINATED TRADEOFFS FROM TRADEOFF ARRAY.
C     ADJUST - ACCEPTS AND EVALUATES MANUAL ADJUSTMENTS TO ROUTE.
C     BKTRAK - BACKTRACKS PROCEDURE TO A PRIOR SOLUTION.
C     DISPLA - DISPLAYS A PRIOR SOLUTION.
C     LOCK   - LOCKS OUT A ROUTE FROM CALCULATIONS OF ROUTE-LOAD
C              DEVIATION AND ROUTE-LENGTH DEVIATION.
C
C
C     IMSL ROUTINES:
C
C        GGUBFS - UNIFORM RANDOM NUMBER GENERATOR.
C        GGUD   - UNIFORM RANDOM VECTOR GENERATOR..
C        VSORA  - MODIFIED QUICKSORT ALGORITHM.
C
C
C                  HIERARCHY OF SUBROUTINE CALLS:
C
C
C     MAIN   CALLS SAVNGS, TWOOPT, THROPT, LDDV2, LDDV3, LNDV2, LNDV3,
C                  NONDOM, TSP, ADJUST, BKTRAK, DISPLA, LOCK  , GGUD,
C                  VSORA
C
C     TWOOPT CALLS FEAS2, XCHNG2, GGUBFS
C     THROPT CALLS FEAS2, XCHNG2, XCHNG3, GGUBFS
```

```
C     LDDV2  CALLS FXCH2, TSP, XCHNG2, GGUBFS
C     LDDV3  CALLS FXCH2, FXCH3, TSP, XCHNG2, XCHNG3, GGUBFS
C     LNDV2  CALLS FXCH2, TSP, XCHNG2, GGUBFS
C     LNDV3  CALLS FXCH2, FXCH3, TSP, XCHNG2, XCHNG3, GGUBFS
C
C
C
C     GRAPHICS CAPABILITY PROVIDED BY TEKTRONIX PLOT 10 TERMINAL
C     CONTROL SYSTEM.
C
C**********************************************************************
C
C
C
C                     VARIABLE DEFINITIONS:
C
C
C     ALLOW      - DROP ALLOWANCE, MEASURED IN DISTANCE UNITS, AT EACH
C                  NODE VISITED.
C     ANSWER     - CHARACTER VARIABLE, 'Y' OR 'N', IN RESPONSE TO
C                  TERMINAL QUERY.
C     BACK       - 0,1 VARIABLE INDICATING BACKWARD TRAVERSAL THROUGH
C                  NETWORK IF VALUE IS 1.
C     BACTRT     - NUMBER OF ACTIVE ROUTES (I.E., ROUTES WITH AT LEAST
C                  1 CUSTOMER) IN THE BEST SOLUTION FOUND THUS FAR IN
C                  MULTIPLE SOLUTIONS.  SEE IACTRT.
C     BCUMLD(I) - CUMULATIVE LOAD AT NODE I IN BEST SOLUTION FOUND THUS
C                  FAR.  SEE CUMLD(I).
C     BCUMLN(I) - CUMULATIVE DISTANCE AT NODE I IN BEST SOLUTION FOUND
C                  THUS FAR.  SEE CUMLN(I).
C     BDEPOT(I) - DEPOT OF ROUTE I IN BEST SOLUTION FOUND THUS FAR.
C                  SEE DEPOT(I).
C     BESTDS     - MINIMUM TOTAL DISTANCE FOUND THUS FAR IN MULTIPLE
C                  SOLUTIONS OF DISTANCE MINIMIZATION PROBLEM.
C     BESTLD(I) - TOTAL LOAD OF ROUTE I IN BEST SOLUTION FOUND THUS
C                  FAR.  SEE LOAD(I).
C     BESTLN(I) - TOTAL LENGTH OF ROUTE I IN BEST SOLUTION FOUND THUS
C                  FAR.  SEE LENGTH(I).
C     BESTRT     - NUMBER OF ROUTES IN BEST SOLUTION FOUND THUS FAR.
C                  SEE IROUTE.
C     BESTP(I)   - PREDECESSOR OF NODE I IN BEST SOLUTION FOUND THUS FAR.
C                  SEE PRED(I).
C     BESTS(I)   - SUCCESSOR OF NODE I IN BEST SOLUTION FOUND THUS FAR.
C                  SEE SUCC(I).
C     BESTTR(I) - ROUTE (TRUCK) ASSIGNED TO NODE I IN BEST SOLUTION
C                  FOUND THUS FAR.  SEE TRUCK(I).
C     BLDDEV     - ROUTE LOAD DEVIATION IN BEST SOLUTION FOUND THUS FAR.
C                  SEE LDDEV.
C     BLNDEV     - ROUTE LENGTH DEVIATION IN BEST SOLUTION FOUND THUS FAR
C                  SEE LNDEV.
C     BMAXLD     - MAXIMUM ROUTE LOAD IN BEST SOLUTION FOUND THUS FAR.
C                  SEE MAXLD.
C     BMINLD     - MINIMUM ROUTE LOAD IN BEST SOLUTION FOUND THUS FAR.
C                  SEE MINLD.
C     BMAXLN     - MAXIMUM ROUTE LENGTH IN BEST SOLUTION FOUND THUS FAR.
C                  SEE MAXLN.
C     BMINLN     - MINIMUM ROUTE LENGTH IN BEST SOLUTION FOUND THUS FAR.
C                    SEE MINLN.
C     BNTRAD     - NUMBER OF ORIGINAL TRADEOFFS FOUND IN FINAL ITERATION
C                  OF BEST SOLUTION FOUND THUS FAR.  SEE NTRADE.
C     BTRADE(,) - ARRAY OF TRADEOFFS FOUND IN FINAL ITERATION OF BEST
C                  SOLUTION FOUND THUS FAR.  SEE TRADE(,).
C     CITY       - NODE INDEX USED IN READING PROBLEM DATA FROM FILE.
C     CLRNDX()   - VECTOR OF COLOR INDEXES USED BY GRAPHICS TERMINAL
C                  PROGRAM.
C     CNSTR1     - CHARACTER VARIABLE CONTAINING DESCRIPTION OF FIRST
C                  CONSTRAINING ACHIEVEMENT LEVEL.
C     CNSTR2     - CHARACTER VARIABLE CONTAINING DESCRIPTION OF SECOND
C                  CONSTRAINING ACHIEVEMENT LEVEL.
C     CUMLD(I)   - CUMULATIVE ROUTE LOAD UP THROUGH NODE I.
C     CUMLN(I)   - CUMULATIVE ROUTE LENGTH UP THROUGH NODE I.
```

```
C       DEMAND(I) - DEMAND AT NODE I.
C       DEPOT(I)  - ARTIFICIAL DEPOT OF ROUTE I.
C       DIST(I,J) - SHORTEST DISTANCE BETWEEN NODES I AND J.
C       DISTLM    - MAXIMUM ROUTE LENGTH ALLOWED IN PROBLEM.
C       DLIMIT    - LIMIT ON TOTAL DISTANCE IN SOLVING A ROUTE-LOAD
C                   DEVIATION OR ROUTE-LENGTH DEVIATION PROBLEM.
C       DSEED     - DOUBLE PRECISION SEED USED IN CALL TO IMSL ROUTINES
C                 - GGUD AND GGUBFS.
C       DSTRLX    - DISTANCE RELAXATION APPLIED TO TOTAL DISTANCE
C                   OBJECTIVE, USED IN GENERATING MULTIPLE FEASIBLE
C                   STARTING SOLUTIONS TO THE THE TOTAL DISTANCE PROBLEM.
C       D1        - DISTANCE BETWEEN POINT1 NODE AND ITS IMMEDIATE
C                   SUCCESSOR.  SIMILAR DEFINITIONS APPLY TO D2 AND D3.
C       END       - LAST NODE IN VRP NETWORK, OR LAST NODE IN TSP PROBLEM.
C       EUCLID    - 0,1 VARIABLE INDICATING WHETHER EUCLIDEAN DISTANCES
C                   ARE TO BE USED IN PROBLEM.  (0=NO, 1=YES).
C       FEAS      - 0,1 VARIABLE INDICATING WHETHER THE ROUTES RESULTING
C                   FROM AN ARC EXCHANGE ARE FEASIBLE WITH RESPECT TO
C                   PROBLEM CONSTRAINTS AND CONSTRAINING GOAL LEVELS.
C       FEASLD(I) - THE POTENTIAL LOAD OF ROUTE I RESULTING FROM AN ARC
C                   EXCHANGE.  USED IN DETERMINING FEASIBILITY OF THE
C                   EXCHANGE.
C       FEASLN(I) - THE POTENTIAL LENGTH OF ROUTE I RESULTING FROM AN
C                   ARC EXCHANGE.  USED IN DETERMINING FEASIBILITY OF THE
C                   EXCHANGE.
C       FEND      - THE POTENTIAL END OF A ROUTE BEING EXAMINED FOR
C                   FEASIBILITY OF THE EXCHANGE.
C       FPRED(I)  - THE POTENTIAL PREDECESSOR OF NODE I RESULTING FROM AN
C                   ARC EXCHANGE.  USED IN DETERMINING FEASIBILITY OF THE
C                   EXCHANGE.
C       FSTART    - THE POTENTIAL START OF A ROUTE BEING EXAMINED FOR
C                   FEASIBILITY OF THE EXCHANGE.
C       FSUCC(I)  - THE POTENTIAL SUCCESSOR OF NODE I RESULTING FROM AN
C                   ARC EXCHANGE.  USED IN DETERMINING FEASIBILITY OF THE
C                   EXCHANGE.
C       HEAD(I)   - BEGINNING NODE IN ROUTE I FORMED BY THE CLARKE AND
C                   WRIGHT SAVINGS ALGORITHM.
C       IACTRT    - THE NUMBER OF ACTIVE ROUTES IN PROBLEM; I.E., ROUTES
C                   HAVING AT LEAST ONE CUSTOMER.
C       IANS      - RESPONSE FROM TERMINAL QUERY.
C       IFLAG(I)  - AN INDICATOR THAT THE ITH VALUE IN THE SAVINGS FILE
C                   HAS BEEN MOVED.  USED IN ALTERING THE SAVINGS FILE FOR
C                   RANDOM STARTING SOLUTIONS.
C       IR(I)     - VECTOR OF SAVINGS VALUES, USED IN ALTERING THE SAVINGS
C                   FILE FOR RANDOM STARTING SOLUTIONS.
C       IROUTE    - NUMBER OF ROUTES IN SOLUTION.
C       IRTN      - STATEMENT NUMBER (BY ASSIGNMENT).
C       ISORT(I)  - SORTED VALUE OF BEGINNING NODE (I) ASSOCIATED WITH
C                   SAVINGS LINK (I,J).
C       ITRADE(,) - VECTOR OF NONDOMINATED TRADEOFFS.  SEE TRADE(,).
C       JSORT(I)  - SORTED VALUE OF ENDING NODE (J) ASSOCIATED WITH
C                   SAVINGS LINK (I,J).
C       LDDEV     - CURRENT VALUE OF ROUTE LOAD DEVIATION.
C       LDDVLM    - LIMIT ON ROUTE LOAD DEVIATION IN SOLVING A TOTAL
C                   DISTANCE OR ROUTE LENGTH DEVIATION PROBLEM.
C       LDRLX     - AMOUNT OF RELAXATION APPLIED TO LDDEV IN FIRST
C                   ITERATION OF ROUTE LOAD DEVIATION ALGORITHM.  USED
C                   FOR FINDING ALTERNATE STARTING SOLUTIONS.
C       LENGTH(I) - DISTANCE TRAVELED OVER ROUTE I.
C       LIMIT1    - AMOUNT OF FIRST CONSTRAINING ACHIEVEMENT LEVEL.  SEE
C                   CNSTR1.
C       LIMIT2    - AMOUNT OF SECOND CONSTRAINING ACHIEVEMENT LEVEL.  SEE
C                   CNSTR2.
C       LOAD(I)   - TOTAL DEMAND CARRIED BY THE ITH VEHICLE.
C       LOKK(I)   - 0,1 VARIABLE DENOTING WHETHER A NODE IS TO BE INCLUDED
C                   IN CALCULATING LNDEV AND LDDEV.  (0=YES, 1=NO).
C       LNDEV     - CURRENT VALUE OF ROUTE-LENGTH DEVIATION.
C       LNDVLM    - LIMIT ON ROUTE-LENGTH DEVIATION IN SOLVING A TOTAL
C                   DISTANCE OR ROUTE-LOAD DEVIATION PROBLEM.
C       LNRLX     - AMOUNT OF RELAXATION APPLIED TO LNDEV IN FIRST
C                   ITERATION OF ROUTE-LENGTH DEVIATION ALGORITHM.  USED
```

```
C                     FOR FINDING ALTERNATE STARTING SOLUTIONS.
C     MAXLD      - AMOUNT OF DEMAND CARRIED BY MOST HEAVILY LOADED
C                     VEHICLE.
C     MAXLN      - DISTANCE DRIVEN OVER LONGEST ROUTE.
C     MENU       - MENU OPTION NUMBER SELECTED.
C     MINLD      - AMOUNT OF DEMAND CARRIED BY LEAST HEAVILY LOADED
C                     VEHICLE.
C     MINLN      - DISTANCE DRIVEN OVER SHORTEST ROUTE.
C     NCITY      - NUMBER OF CUSTOMERS IN PROBLEM.
C     NT         - NUMBER OF NONDOMINATED (REDUCED) TRADEOFFS.
C     NTRADE     - NUMBER OF ORIGINAL TRADEOFFS.
C     NULL       - 0,1 VARIABLE INDICATING WHETHER AN EXCHANGE IS NULL
C                     (0=NO, 1=YES).
C     OBJ        - CHARACTER VARIABLE CONTAINING NAME OF CURRENT
C                     OBJECTIVE FUNCTION BEING MINIMIZED.
C     OLDOBJ     - NUMBER OF OBJECTIVE FUNCTION MINIMIZED IN PREVIOUS
C                     ITERATION OF PROCEDURE.
C     PERMI(I)   - VARIABLE TO HOLD BEGINNING NODE OF ITH SAVINGS LINK.
C                     USED TO RESTORE SAVINGS FILE BACK TO ITS ORIGINAL
C                     SORTED ORDER AFTER A RANDOM PERTURBATION OF THE FILE
C                     HAS BEEN MADE.
C     PERMJ(I)   - VARIABLE TO HOLD ENDING NODE OF ITH SAVINGS LINK.
C                     CORRESPONDS TO PERMI(I).
C     PERMSV(I) - VARIABLE TO HOLD THE ITH SAVINGS VALUE.  USED TO
C                     RESTORE SAVINGS FILE TO ITS ORIGINAL SORTED ORDER
C                     AFTER A RANDOM PERTURBATION OF THE FILE HAS BEEN MADE
C     PNAME      - CHARACTER VARIABLE CONTAINING PROBLEM IDENTIFICATION.
C     POINT1     - BEGINNING NODE OF FIRST ARC TO BE REPLACED IN AN ARC
C                     EXCHANGE.
C     POINT2     - BEGINNING NODE OF SECOND ARC TO BE REPLACED IN AN ARC
C                     EXCHANGE.
C     POINT3     - BEGINNING NODE OF THIRD ARC TO BE REPLACED IN AN ARC
C                     EXCHANGE.
C     PRED(I)    - PREDECESSOR OF THE ITH NODE.
C     RTSIZE(I) - NUMBER OF CUSTOMERS SERVED ON THE ITH ROUTE.
C     SAVING(I,J)-SAVINGS CRITERION FOR LINKING NODES I AND J.
C     SOLNO      - SOLUTION NUMBER.
C     SORT(I)    - ITH SORTED SAVINGS VALUE.
C     START      - BEGINNING NODE OF A TSP, OR BEGINNING NODE OF AN ARC
C                     EXCHANGE ALGORITHM.
C     STCULD(I,J)-VALUE OF CUMLD(J) STORED FOR THE ITH SOLUTION.
C     STCULN(I,J)-VALUE OF CUMLN(J) STORED FOR THE ITH SOLUTION.
C     STDLIM     - VALUE OF DLIMIT STORED FOR THE ITH SOLUTION.
C     STDEP(I,J)- VALUE OF DEPOT(J) STORED FOR THE ITH SOLUTION.
C     STITRD(I,,)-VALUE OF ITRADE(,,) STORED FOR THE ITH SOLUTION.
C     STLDVL(I) - VALUE OF LDDVLM STORED FOR THE ITH SOLUTION.
C     STLNTH(I,J)-VALUE OF LENGTH(J) STORED FOR THE ITH SOLUTION.
C     STLNVL(I) - VALUE OF LNDVLM STORED FOR THE ITH SOLUTION.
C     STLOAD(I,J)-VALUE OF LOAD(J) STORED FOR THE ITH SOLUTION.
C     STLOKK(I,J)-VALUE OF LOKK(J) STORED FOR THE ITH SOLUTION.
C     STMNLD(I) - VALUE OF MINLD STORED FOR THE ITH SOLUTION.
C     STMNLN(I) - VALUE OF MINLN STORED FOR THE ITH SOLUTION.
C     STMXLD(I) - VALUE OF MAXLD STORED FOR THE ITH SOLUTION.
C     STMXLN(I) - VALUE OF MAXLN STORED FOR THE ITH SOLUTION.
C     STNT(I)    - VALUE OF NT STORED FOR THE ITH SOLUTION.
C     STOBJ(I)   - VALUE OF OBJ STORED FOR THE ITH SOLUTION.
C     STPRED(I,J)-VALUE OF PRED(J) STORED FOR THE ITH SOLUTION.
C     STSUCC(I,J)-VALUE OF SUCC(J) STORED FOR THE ITH SOLUTION.
C     STTRK(I,J) -VALUE OF TRUCK(J) STORED FOR THE ITH SOLUTION.
C     SUCC(I)    - SUCCESSOR OF THE ITH NODE.
C     TAIL(I)    - LAST NODE IN ROUTE I FORMED BY CLARKE AND WRIGHT
C                     SAVINGS ALGORITHM.
C     TARRAY(I) - ARRAY FOR TERMINAL CONTROL VIA PLOT 10 PACKAGE.
C     TDIST      - TOTAL DISTANCE OVER ALL ROUTES.
C     TIME       - ELAPSED CPU TIME.
C     TRADE(,)   - ARRAY OF ORIGINAL TRADEOFFS.  SEE ITRADE(,).
C     TRUCK(I)   - VEHICLE SERVING ITH CUSTOMER.
C     TRY(I,J)   - 0,1 VARIABLE INDICATING WHETHER A 2-ARC EXCHANGE HAS
C                     ALREADY BEEN EVALUATED FOR THE TWO ARCS BEGINNING
C                     WITH NODES I AND J.  USED TO PREVENT MULTIPLE
C                     EVALUATIONS OF THE SAME EXCHANGE.
```

```
C     WK(I)      - WORK VECTOR USED IN CALL TO IMSL ROUTINE VSORA.
C     WORK(I)    - WORK VECTOR USED IN CALL TO IMSL ROUTINE VSORA.
C     WTLIM      - VEHICLE CAPACITY (WEIGHT LIMIT).
C     XCENTR(I) - HORIZONTAL COMPONENT OF CENTROID OF ITH ROUTE.
C     XCOORD(I) - HORIZONTAL COORDINATE OF NODE I.
C     YCENTR(I) - VERTICAL COMPONENT OF CENTROID OF ITH ROUTE.
C     YCOORD(I) - VERTICAL COORDINATE OF NODE I.
C
C
C***********************************************************************
C
C         MAIN PROGRAM
C
C***********************************************************************
C
C
      CHARACTER*44 PNAME,IPLACE
      CHARACTER*16 OBJ,STOBJ(12),CNSTR1,CNSTR2
      CHARACTER*1 ANSWER
      INTEGER EUCLID,CITY,XCOORD(0:120),YCOORD(0:120),DEMAND(0:120)
      INTEGER HEAD(120),TAIL(120),PRED(120),SUCC(120),ROUTES,TWGT,WTLIM
      INTEGER DIST,ALLOW,TDIST,DISTLM,TRUCK,IR(100),IFLAG(40),
     * PERMI(40),PERMJ(40)
      DOUBLE PRECISION DSEED
      INTEGER START,END,POINT1,POINT2,D,FEASLD(20),FEASLN(20)
      INTEGER FTRUCK,FWD,BACK,TEMTRK(120),CUMLD(120),CUMLN(120)
      INTEGER FOUND,TRCNT,TRK,TAG,D1,D2,D3,D4,D5,D6,D7,D8
      INTEGER FSTART,FEND,FPRED(120),FSUCC(120),BACTRT,DSTRLX
      INTEGER PERMPR(120),PERMSU(120),PERMTR(120),RTSIZE(20)
      INTEGER BESTRT,BESTDS,BCUMLN(120),BCUMLD(120),BESTP(120)
      INTEGER BESTS(120),BESTLN(20),BESTLD(20),BDEPOT(20),BESTTR(120)
      INTEGER DLIMIT,TARRAY(40),DEPOT(20),CLRNDX(12),BLDDEV,BLNDEV
      INTEGER BMAXLD,BMINLD,BMAXLN,BMINLN,LOKK(120),STLOKK(12,120)
      INTEGER BNTRAD,ITRADE(7,500),OLDOBJ
      INTEGER SOLNO,STPRED(12,120),STSUCC(12,120),STTRK(12,120),
     *STCULN(12,120),STCULD(12,120),STDEP(12,20),STLOAD(12,20),
     *STLNTH(12,20),STMXLN(12),STMNLN(12),STMXLD(12),STMNLD(12),
     *STITRD(12,7,500),STLDVL(12),STLNVL(12),STDLIM(12),STNT(12)
      DIMENSION BTRADE(7,500),TRADE(7,500),WK(14)
      DIMENSION DIST(0:120,0:120),SAVING(3,6000),SORT(6000),PERMSV(40)
      DIMENSION ISORT(6000),JSORT(6000),LOAD(120),TRUCK(120)
      DIMENSION LENGTH(120),WORK(6),XCENTR(20),YCENTR(20)
      COMMON IROUTE,NCITY,IRUN,START,END,WTLIM,DISTLM,ALLOW,NPERM,
     *ICOUNT,LDDVLM,LNDVLM,DLIMIT,MAXLD,MINLD,MAXLN,MINLN,D1,D2,D3,D4,
     *D5,D6,DEPOT,PRED,SUCC,TRUCK,DEMAND,LENGTH,LOAD,CUMLD,CUMLN,TEMTRK,
     *FEASLD,FEASLN,PERMPR,PERMSU,PERMTR,ISORT,JSORT,SORT,DIST,XCOORD,
     *YCOORD,LOKK,TRADE,NTRADE,DSEED
      COMMON /STORE/STPRED,STSUCC,STTRK,STCULN,STCULD,STDEP,STLOAD,
     *STLNTH,STMXLN,STMNLN,STMXLD,STMNLD,STITRD,STNT,STLDVL,STLNVL,
     *STDLIM,STLOKK,STOBJ
C
C
C
C
C
C         INITIALIZE TIMER
C
C
      TIME=0.0
      NTRADE=0
      NT=0
      CALL ELAPSE(ITIME)
      CALL ELAPSE(ITIME)
      LDDVLM=99000
      LNDVLM=99000
C
C
C
C
C     SET UP THE GRAPHICS SCREEN
C
```

```
C
C
      CALL INITT(480)
      CALL TERM(3,4096)
      CALL CHRSIZ(1)
      TARRAY(1)=27
      TARRAY(2)=75
      TARRAY(3)=65
      TARRAY(4)=48
      TARRAY(2)=77
      TARRAY(3)=76
      TARRAY(4)=50
      CALL MOVABS(1030,0)
      CALL DRWABS(4095,0)
      CALL DRWABS(4095,3120)
      CALL DRWABS(1030,3120)
      CALL DRWABS(1030,0)
      CALL MOVABS(1030,2800)
      CALL DRWABS(4095,2800)
      CALL MOVABS(1030,3120)
      CALL DRWABS(0,3120)
      CALL DRWABS(0,0)
      CALL DRWABS(1030,0)
      CALL MOVABS(1130,2950)
      TARRAY(2)=77
      TARRAY(3)=84
      TARRAY(4)=53
      CALL AOUTST(44,PNAME)
C
C
C
C
C
C         READ IN THE PROBLEM DATA
C
C
      CALL ANMODE
      READ(8,100) PNAME
      READ(8,101) NCITY,EUCLID,WTLIM,DISTLM,ALLOW
      MAXX=-999999
      MAXY=-999999
      MINX=999999
      MINY=999999
      DO 1 I=0,NCITY
        READ(8,102) CITY,XCOORD(CITY),YCOORD(CITY),DEMAND(CITY)
        IF(XCOORD(CITY).GT.MAXX) MAXX=XCOORD(CITY)
        IF(XCOORD(CITY).LT.MINX) MINX=XCOORD(CITY)
        IF(YCOORD(CITY).GT.MAXY) MAXY=YCOORD(CITY)
        IF(YCOORD(CITY).LT.MINY) MINY=YCOORD(CITY)
    1 CONTINUE
      LIM=MAX0(MAXX-MINX,MAXY-MINY)
      X1=MINX-10
      X2=X1+FLOAT(LIM)+20.
      Y1=MINY-10
      Y2=Y1+FLOAT(LIM)+20.
      CALL DWINDO(X1,X2,Y1,Y2)
C
C
C         IF THE PROBLEM HAS EUCLIDEAN DISTANCES, CALCULATE THEM.
C
C
      IF(EUCLID.EQ.1) THEN
        DO 2 I=0,NCITY-1
        DO 2 J=I+1,NCITY
          DIST(I,J)=SQRT(FLOAT((XCOORD(I)-XCOORD(J))**2+(YCOORD(I)
     *                    - YCOORD(J))**2)) + 0.5
    2   DIST(J,I)=DIST(I,J)
        GOTO 4
      END IF
C
C
```

```
C         IF PROBLEM HAS NON-EUCLIDEAN DISTANCES, READ THEM IN.
C
C
      IF(EUCLID.EQ.O) THEN
    3   READ(8,103,END=4) I,J,DIST(I,J)
        DIST(J,I)=DIST(I,J)
        GOTO 3
      END IF
    4 CONTINUE
C
C
C         CALCULATE THE SAVINGS FILE.
C
C
      DSEED=6.ODO
      ICOUNT=O
      DO 6 I=1,NCITY-1
      DO 6 J=I+1,NCITY
        ICOUNT=ICOUNT+1
        SAVING(1,ICOUNT)=FLOAT(DIST(O,I)+DIST(J,O)-DIST(I,J))*(-1.)
        SAVING(2,ICOUNT)=I
    6   SAVING(3,ICOUNT)=J
C
C
C         SORT THE SAVINGS FILE -- IMSL MODIFIED QUICKSORT SUBROUTINE
C
C
C
      CALL VSORA(SAVING,3,3,ICOUNT,1,WORK,IER)
      DO 7 I=1,ICOUNT
        SORT(I)=-SAVING(1,I)
        ISORT(I)=SAVING(2,I)
    7   JSORT(I)=SAVING(3,I)
      DO 8 I=1,40
        PERMSV(I)=SORT(I)
        PERMI(I)=ISORT(I)
    8   PERMJ(I)=JSORT(I)
C
C
      DO 9 I=1,120
    9 LOKK(I)=O
      DO 10 I=1,7
      DO 10 J=1,500
   10 TRADE(I,J)=O.
C
C
C
      DO 70 IRUN=1,8
C
C
C         ALTER THE SAVINGS FILE FOR A NEW INITIAL RANDOM SOLUTION
C
C
C
      IF(IRUN.GT.1) THEN
        CALL GGUD(DSEED,IROUTE*2,2*(IROUTE*2),IR)
        DO 12 I=1,40
   12     IFLAG(I)=O
        MOVES=O
        DO 13 I=1,2*(IROUTE*2)
          IF(IFLAG(IR(I)).GT.O) GOTO 13
C         ELSE
          MOVES=MOVES+1
          SORT(MOVES)=PERMSV(IR(I))
          ISORT(MOVES)=PERMI(IR(I))
          JSORT(MOVES)=PERMJ(IR(I))
          IFLAG(IR(I))=1
          IF(MOVES.GE.IROUTE*2) GOTO 14
   13   CONTINUE
      END IF
   14 CONTINUE
```

```
C
C
C
      CALL SAVNGS
C
C
C         RESTORE SAVINGS FILE TO ORIGINAL SORTED ORDER
C
C
      DO 15 I=1,40
        SORT(I)=PERMSV(I)
        ISORT(I)=PERMI(I)
   15   JSORT(I)=PERMJ(I)
C
C
C
   29 FWD=1
      BACK=2
      START=NCITY+1
      END=PRED(START)
      INSIDE=0
      NEXT=NCITY+1
   31 NODE=NEXT
      IF(NODE.EQ.NCITY+1.AND.INSIDE.EQ.1) GOTO 32
      INSIDE=1
      IF(NODE.GT.NCITY) THEN
        ILN=0
        ILD=0
        CUMLD(NODE)=0
        CUMLN(NODE)=0
      ENDIF
      ILD=ILD+DEMAND(NODE)
      CUMLD(NODE)=ILD
      IF(NODE.LE.NCITY) ILN=ILN+DIST(NODE,PRED(NODE))+ALLOW
      CUMLN(NODE)=ILN
      NEXT=SUCC(NODE)
      GOTO 31
   32 CONTINUE
      CALL TWOOPT(0)
      CALL THROPT(0)
      LOADT=0
      LENGT=0
      DO 33 I=1,IROUTE
        LOADT=LOADT+LOAD(I)
   33   LENGT=LENGT+LENGTH(I)
C
C
C
C     SAVE BEST SOLUTION.
C
C
      IF(IRUN.EQ.1) THEN
        IACTRT=0
        DO 59 I=1,IROUTE
          IF(LOAD(I).GT.0) IACTRT=IACTRT+1
   59  CONTINUE
       BACTRT=IACTRT
        BESTRT=IROUTE
        BESTDS=LENGT
        DO 63 I=1,NCITY+IROUTE
          BESTP(I)=PRED(I)
          BESTS(I)=SUCC(I)
          BESTTR(I)=TRUCK(I)
          BCUMLN(I)=CUMLN(I)
   63     BCUMLD(I)=CUMLD(I)
        DO 64 I=1,IROUTE
          BESTLD(I)=LOAD(I)
          BESTLN(I)=LENGTH(I)
   64     BDEPOT(I)=DEPOT(I)
      GOTO 70
      ENDIF
```

```
      IACTRT=0
      DO 65 I=1,IROUTE
        IF(LOAD(I).GT.0) IACTRT=IACTRT+1
   65 CONTINUE
      IF(IACTRT.GT.BACTRT) GOTO 70
      IF(IACTRT.EQ.BACTRT.AND.LENGT.GE.BESTDS) GOTO 70
C     ELSE
      BESTRT=IROUTE
      BESTDS=LENGT
      BACTRT=IACTRT
      DO 66 I=1,NCITY+IROUTE
        BESTP(I)=PRED(I)
        BESTS(I)=SUCC(I)
        BESTTR(I)=TRUCK(I)
        BCUMLN(I)=CUMLN(I)
   66   BCUMLD(I)=CUMLD(I)
      DO 67 I=1,IROUTE
        BESTLD(I)=LOAD(I)
        BESTLN(I)=LENGTH(I)
   67   BDEPOT(I)=DEPOT(I)
   70 CONTINUE
C
C
C
C     RECALL BEST SOLUTION.
C
C
      IROUTE=BESTRT
      LENGT=BESTDS
      DO 71 I=1,NCITY+IROUTE
        PRED(I)=BESTP(I)
        SUCC(I)=BESTS(I)
        CUMLN(I)=BCUMLN(I)
        CUMLD(I)=BCUMLD(I)
   71   TRUCK(I)=BESTTR(I)
      TDIST=0
      DO 72 I=1,IROUTE
        RTSIZE(I)=0
        XCENTR(I)=0.0
        YCENTR(I)=0.0
        LOAD(I)=BESTLD(I)
        LENGTH(I)=BESTLN(I)
        TDIST=TDIST+LENGTH(I)
   72   DEPOT(I)=BDEPOT(I)
      CALL TWINDO(1010,4095,0,2800)
      CALL CHRSIZ(3)
      NEXT=NCITY+1
      X=XCOORD(NCITY+1)
      Y=YCOORD(NCITY+1)
      XCENTR(TRUCK(NCITY+1))=XCENTR(TRUCK(NCITY+1)) + X
      YCENTR(TRUCK(NCITY+1))=YCENTR(TRUCK(NCITY+1)) + Y
      RTSIZE(TRUCK(NCITY+1))=RTSIZE(TRUCK(NCITY+1)) + 1
      CALL MOVEA(X,Y)
      CALL MOVREL(40,0)
      CALL DRWREL(0,40)
      CALL DRWREL(-80,0)
      CALL DRWREL(0,-80)
      CALL DRWREL(80,0)
      CALL DRWREL(0,40)
      CALL MOVEA(X,Y)
      INSIDE=0
   73 NODE=NEXT
      IF(NODE.EQ.NCITY+1.AND.INSIDE.EQ.1) GOTO 74
      INSIDE=1
      X=XCOORD(NODE)
      Y=YCOORD(NODE)
      XCENTR(TRUCK(NODE))=XCENTR(TRUCK(NODE)) + X
      YCENTR(TRUCK(NODE))=YCENTR(TRUCK(NODE)) + Y
      RTSIZE(TRUCK(NODE))=RTSIZE(TRUCK(NODE)) + 1
      CALL DRAWA(X,Y)
      ICHR1=NODE/100
```

```
      ICHR2=NODE/10 - ICHR1*10
      ICHR3=NODE-(ICHR1*100 + ICHR2*10)
      IF(NODE.LE.NCITY) THEN
        CALL MOVREL(20,0)
        CALL DRWREL(0,20)
        CALL DRWREL(-40,0)
        CALL DRWREL(0,-40)
        CALL DRWREL(40,0)
        CALL DRWREL(0,20)
      ENDIF
      CALL MOVEA(X,Y)
      NEXT=SUCC(NODE)
      GOTO 73
   74 X=XCOORD(NODE)
      Y=YCOORD(NODE)
      CALL DRAWA(X,Y)
      CALL CHRSIZ(1)
      DO 744 J=1,IROUTE
        IF(LOAD(J).GT.0) THEN
        XCENTR(J)=XCENTR(J)/FLOAT(RTSIZE(J))
        YCENTR(J)=YCENTR(J)/FLOAT(RTSIZE(J))
        CALL MOVEA(XCENTR(J),YCENTR(J))
        ICHR1=J/10
        ICHR2=J-ICHR1*10
        IF(ICHR1.NE.0) CALL ANCHO(ICHR1+48)
        CALL ANCHO(ICHR2+48)
        ENDIF
  744 CONTINUE
      CALL CHRSIZ(1)
      IMAXLN=-99
      IMAXLD=-99
      IMINLN=999999
      IMINLD=999999
      DO 75 I=1,IROUTE
        IF(LOKK(DEPOT(I)).NE.0) GOTO 75
        IF(LENGTH(I).GT.IMAXLN) IMAXLN=LENGTH(I)
        IF(LENGTH(I).LT.IMINLN.AND.LENGTH(I).GT.0) IMINLN=LENGTH(I)
        IF(LOAD(I).GT.IMAXLD) IMAXLD=LOAD(I)
        IF(LOAD(I).LT.IMINLD.AND.LOAD(I).GT.0) IMINLD=LOAD(I)
   75  CONTINUE
      MAXLN=IMAXLN
      MINLN=IMINLN
      MAXLD=IMAXLD
      MINLD=IMINLD
      LDDEV=MAXLD-MINLD
      LNDEV=MAXLN-MINLN
C
C
C
C     WRITE OUT THE BEGINNING ROUTE SET.
C
C
      SOLNO=1
      CALL MOVABS(1030,2950)
      CALL AOUTST(44,PNAME)
      CALL TWINDO(0,4095,0,3120)
      CALL HOME
      CALL ANMODE
      WRITE(6,319)
  319 FORMAT(1H ,//'  SOLUTION NUMBER 1'//' BEGINNING ROUTE SET'
     *//' ROUTE  LOAD  LENGTH'/)
      DO 76 II=1,IROUTE
      IF(LOAD(II).GT.0) WRITE(6,321) II,LOAD(II),LENGTH(II)
   76 CONTINUE
  321 FORMAT(I4,I8,1X,I5)
      WRITE(6,322) TDIST,LDDEV,LNDEV
  322 FORMAT(' TOT. DIST =',I6,/,' LOAD DEV. = ',I5,/,' LENGTH DEV. =',I
     *4)
      CALL ELAPSE(ITIME)
      TIME=TIME+FLOAT(ITIME)/1000.
      WRITE(6,323) TIME
```

```
  323 FORMAT(1H ,'CPU SECONDS:',F6.2)
      WRITE(6,400)
      READ(5,100) ANSWER
      IF(ANSWER.EQ.'Y') THEN
        CALL FINITT(0,700)
        STOP
      ENDIF
C
C
C
C     STORE THE BEGINNING SOLUTION
C
C
      STDLIM(SOLNO)=DLIMIT
      STLDVL(SOLNO)=LDDVLM
      STLNVL(SOLNO)=LNDVLM
      STMNLD(SOLNO)=MINLD
      STMXLD(SOLNO)=MAXLD
      STMNLN(SOLNO)=MINLN
      STMXLN(SOLNO)=MAXLN
      STNT(SOLNO)=0
      STOBJ(SOLNO)='TOTAL DISTANCE'
      DO 78 J=1,IROUTE
        STDEP(SOLNO,J)=DEPOT(J)
        STLNTH(SOLNO,J)=LENGTH(J)
   78   STLOAD(SOLNO,J)=LOAD(J)
      DO 79 J=1,NCITY+IROUTE
        STCULD(SOLNO,J)=CUMLD(J)
        STCULN(SOLNO,J)=CUMLN(J)
        STLOKK(SOLNO,J)=LOKK(J)
        STPRED(SOLNO,J)=PRED(J)
        STSUCC(SOLNO,J)=SUCC(J)
   79   STTRK(SOLNO,J)=TRUCK(J)
C
C
C     ELSE
      WRITE(6,402)
      READ(5,*) IANS
C
C
C     SOLVE A ROUTE-LENGTH DEVIATION PROBLEM
C
      IF(IANS.EQ.1) THEN
        OBJ='LENGTH DEVIATION'
        OLDOBJ=3
        DLIMIT=TDIST
        LDDVLM=LDDEV
        CNSTR1='TOTAL DISTANCE'
        CNSTR2='LOAD DEVIATION'
        CALL NEWPAG
        WRITE(6,404) OBJ,CNSTR1,DLIMIT,CNSTR2,LDDVLM
        LIMIT1=DLIMIT
        LIMIT2=LDDVLM
        NT=0
        CALL LNDV2(0)
        CALL LNDV3(0)
        BNTRAD=NTRADE
        DO 77 I=1,NTRADE
        DO 77 J=1,7
   77     BTRADE(J,I)=TRADE(J,I)
        DO 80 I=1,IROUTE
          BESTLD(I)=LOAD(I)
          BESTLN(I)=LENGTH(I)
   80     BDEPOT(I)=DEPOT(I)
        DO 81 I=1,NCITY+IROUTE
          BESTP(I)=PRED(I)
          BESTS(I)=SUCC(I)
          BESTTR(I)=TRUCK(I)
          BCUMLN(I)=CUMLN(I)
   81     BCUMLD(I)=CUMLD(I)
        LNDEV=MAXLN-MINLN
```

```
      BMAXLD=MAXLD
      BMINLD=MINLD
      BMAXLN=MAXLN
      BMINLN=MINLN
      BLNDEV=LNDEV
      LNRLX=FLOAT(LNDEV)*0.5
      CALL LNDV3(LNRLX)
      LNDEV=MAXLN-MINLN
      IF(LNDEV.GE.BLNDEV) THEN
        LNDEV=BLNDEV
        MAXLD=BMAXLD
        MINLD=BMINLD
        MAXLN=BMAXLN
        MINLN=BMINLN
        DO 83 I=1,IROUTE
          LOAD(I)=BESTLD(I)
          LENGTH(I)=BESTLN(I)
 83       DEPOT(I)=BDEPOT(I)
        DO 84 I=1,NCITY+IROUTE
          PRED(I)=BESTP(I)
          SUCC(I)=BESTS(I)
          TRUCK(I)=BESTTR(I)
          CUMLN(I)=BCUMLN(I)
 84       CUMLD(I)=BCUMLD(I)
        NTRADE=BNTRAD
        DO 82 I=1,NTRADE
        DO 82 J=1,7
 82       TRADE(J,I)=BTRADE(J,I)
      ENDIF
      IF(NTRADE.GT.0) CALL VSORA(TRADE,7,7,NTRADE,1,WK,IER)
      NT=0
      IF(NTRADE.EQ.0) GOTO 93
C     ELSE
      PREV1=999999.
      PREV2=-999999.
      PREV3=-999999.
      DO 823 I=1,NTRADE
        IF(TRADE(1,I).NE.PREV1) THEN
          NT=NT+1
          ITRADE(1,NT)=TRADE(1,I)
          ITRADE(2,NT)=TRADE(2,I)
          ITRADE(3,NT)=TRADE(3,I)
          ITRADE(4,NT)=TRADE(4,I)
          ITRADE(5,NT)=TRADE(5,I)
          ITRADE(6,NT)=TRADE(6,I)
          ITRADE(7,NT)=TRADE(7,I)
          PREV1=TRADE(1,I)
          PREV2=TRADE(2,I)
          PREV3=TRADE(3,I)
          GOTO 823
        ENDIF
        IF(TRADE(2,I).LE.PREV2.AND.TRADE(3,I).LE.PREV3) THEN
          ITRADE(2,NT)=TRADE(2,I)
          ITRADE(3,NT)=TRADE(3,I)
          ITRADE(4,NT)=TRADE(4,I)
          ITRADE(5,NT)=TRADE(5,I)
          ITRADE(6,NT)=TRADE(6,I)
          ITRADE(7,NT)=TRADE(7,I)
          PREV1=TRADE(1,I)
          PREV2=TRADE(2,I)
          PREV3=TRADE(3,I)
        ENDIF
 823  CONTINUE
      CALL NONDOM(ITRADE,NT)
      GOTO 93
    ENDIF
C
C
C   SOLVE A ROUTE-LOAD DEVIATION PROBLEM.
C
C
```

```
      IF(IANS.EQ.2) THEN
        OBJ='LOAD DEVIATION'
        OLDOBJ=2
        DLIMIT=TDIST
        LNDVLM=LNDEV
        CNSTR1='TOTAL DISTANCE'
        CNSTR2='LENGTH DEVIATION'
        CALL NEWPAG
        WRITE(6,404) OBJ,CNSTR1,DLIMIT,CNSTR2,LNDVLM
        LIMIT1=DLIMIT
        LIMIT2=LNDVLM
        NT=0
        DO 901 IRUN=1,3
          LDRLX=FLOAT(IRUN-1)*FLOAT(LDDEV)*0.5
          IF(IRUN.EQ.1) CALL LDDV2(0)
          CALL LDDV3(LDRLX)
          LDDEV=MAXLD-MINLD
          IF(IRUN.EQ.1.OR.LDDEV.LT.BLDDEV) THEN
            BNTRAD=NTRADE
            DO 85 I=1,NTRADE
            DO 85 J=1,7
85            BTRADE(J,I)=TRADE(J,I)
            DO 87 I=1,IROUTE
              BESTLD(I)=LOAD(I)
              BESTLN(I)=LENGTH(I)
87            BDEPOT(I)=DEPOT(I)
            DO 88 I=1,NCITY+IROUTE
              BESTP(I)=PRED(I)
              BESTS(I)=SUCC(I)
              BESTTR(I)=TRUCK(I)
              BCUMLN(I)=CUMLN(I)
88            BCUMLD(I)=CUMLD(I)
            BMAXLD=MAXLD
            BMINLD=MINLD
            BMAXLN=MAXLN
            BMINLN=MINLN
            BLDDEV=LDDEV
          ENDIF
901     CONTINUE
          LDDEV=BLDDEV
          MAXLD=BMAXLD
          MINLD=BMINLD
          MAXLN=BMAXLN
          MINLN=BMINLN
          DO 89 I=1,IROUTE
            LOAD(I)=BESTLD(I)
            LENGTH(I)=BESTLN(I)
89          DEPOT(I)=BDEPOT(I)
          DO 90 I=1,NCITY+IROUTE
            PRED(I)=BESTP(I)
            SUCC(I)=BESTS(I)
            TRUCK(I)=BESTTR(I)
            CUMLN(I)=BCUMLN(I)
90          CUMLD(I)=BCUMLD(I)
          NTRADE=BNTRAD
          DO 91 I=1,NTRADE
          DO 91 J=1,7
91          TRADE(J,I)=BTRADE(J,I)
        IF(NTRADE.GT.0) CALL VSORA(TRADE,7,7,NTRADE,1,WK,IER)
        NT=0
        IF(NTRADE.EQ.0) GOTO 93
C       ELSE
        PREV1=999999.
        PREV2=-999999.
        PREV3=-999999.
        DO 826 I=1,NTRADE
          IF(TRADE(1,I).NE.PREV1) THEN
            NT=NT+1
            ITRADE(1,NT)=TRADE(1,I)
            ITRADE(2,NT)=TRADE(2,I)
            ITRADE(3,NT)=TRADE(3,I)
```

```
          ITRADE(4,NT)=TRADE(4,I)
          ITRADE(5,NT)=TRADE(5,I)
          ITRADE(6,NT)=TRADE(6,I)
          ITRADE(7,NT)=TRADE(7,I)
          PREV1=TRADE(1,I)
          PREV2=TRADE(2,I)
          PREV3=TRADE(3,I)
          GOTO 826
        ENDIF
        IF(TRADE(2,I).LE.PREV2.AND.TRADE(3,I).LE.PREV3) THEN
          ITRADE(2,NT)=TRADE(2,I)
          ITRADE(3,NT)=TRADE(3,I)
          ITRADE(4,NT)=TRADE(4,I)
          ITRADE(5,NT)=TRADE(5,I)
          ITRADE(6,NT)=TRADE(6,I)
          ITRADE(7,NT)=TRADE(7,I)
          PREV1=TRADE(1,I)
          PREV2=TRADE(2,I)
          PREV3=TRADE(3,I)
        ENDIF
  826   CONTINUE
        CALL NONDOM(ITRADE,NT)
        GOTO 93
      ENDIF
C
C
C
C     DISPLAY SOLUTION ON GRAPHICS SCREEN
C
C
C
   93 SOLNO=SOLNO+1
      NEXT=NCITY+1
      CALL NEWPAG
      CALL TWINDO(0,4095,0,3120)
      CALL MOVABS(0,0)
      CALL DRWABS(4095,0)
      CALL DRWABS(4095,3120)
      CALL DRWABS(0,3120)
      CALL DRWABS(0,0)
      CALL MOVABS(1030,0)
      CALL DRWABS(1030,3120)
      CALL MOVABS(1030,2800)
      CALL DRWABS(4095,2800)
      CALL MOVABS(1130,2950)
      CALL AOUTST(44,PNAME)
      CALL TWINDO(1030,4095,0,2800)
      X1=MINX-10
      X2=X1+FLOAT(LIM)+20.
      Y1=MINY-10
      Y2=Y1+FLOAT(LIM)+20.
      CALL DWINDO(X1,X2,Y1,Y2)
      CALL CHRSIZ(3)
      DO 94 J=1,IROUTE
        RTSIZE(J)=0
        XCENTR(J)=0.0
   94   YCENTR(J)=0.0
      X=XCOORD(NEXT)
      Y=YCOORD(NEXT)
      XCENTR(TRUCK(NEXT))=XCENTR(TRUCK(NEXT)) + X
      YCENTR(TRUCK(NEXT))=YCENTR(TRUCK(NEXT)) + Y
      RTSIZE(TRUCK(NEXT))=RTSIZE(TRUCK(NEXT)) + 1
      CALL MOVEA(X,Y)
      CALL MOVREL(40,0)
      CALL DRWREL(0,40)
      CALL DRWREL(-80,0)
      CALL DRWREL(0,-80)
      CALL DRWREL(80,0)
      CALL DRWREL(0,40)
      CALL MOVEA(X,Y)
      INSIDE=0
```

```
   95 NODE=NEXT
      IF(NODE.EQ.NCITY+1.AND.INSIDE.EQ.1) GOTO 96
      INSIDE=1
      X=XCOORD(NODE)
      Y=YCOORD(NODE)
      XCENTR(TRUCK(NODE))=XCENTR(TRUCK(NODE)) + X
      YCENTR(TRUCK(NODE))=YCENTR(TRUCK(NODE)) + Y
      RTSIZE(TRUCK(NODE))=RTSIZE(TRUCK(NODE)) + 1
      CALL DRAWA(X,Y)
      ICHR1=NODE/100
      ICHR2=NODE/10 - ICHR1*10
      ICHR3=NODE - (ICHR1*100 + ICHR2*10)
      IF(NODE.LE.NCITY) THEN
        CALL MOVREL(20,0)
        CALL DRWREL(0,20)
        CALL DRWREL(-40,0)
        CALL DRWREL(0,-40)
        CALL DRWREL(40,0)
        CALL DRWREL(0,20)
      ENDIF
      CALL MOVEA(X,Y)
      NEXT=SUCC(NODE)
      GOTO 95
   96 CONTINUE
      X=XCOORD(NODE)
      Y=YCOORD(NODE)
      CALL DRAWA(X,Y)
      CALL CHRSIZ(1)
      DO 97 J=1,IROUTE
        IF(LOAD(J).GT.0) THEN
        XCENTR(J)=XCENTR(J)/FLOAT(RTSIZE(J))
        YCENTR(J)=YCENTR(J)/FLOAT(RTSIZE(J))
        CALL MOVEA(XCENTR(J),YCENTR(J))
        ICHR1=J/10
        ICHR2=J-ICHR1*10
        IF(ICHR1.NE.0) CALL ANCHO(ICHR1+48)
        CALL ANCHO(ICHR2+48)
        ENDIF
   97 CONTINUE
      CALL CHRSIZ(1)
      TDIST=0
      DO 98 I=1,IROUTE
   98 TDIST=TDIST+LENGTH(I)
      LNDEV=MAXLN-MINLN
      LDDEV=MAXLD-MINLD
      CALL HOME
      CALL ANMODE
      WRITE(6,405) SOLNO,OBJ
      DO 99 II=1,IROUTE
      IF(LOAD(II).LE.0) GOTO 99
      WRITE(6,321) II,LOAD(II),LENGTH(II)
   99 CONTINUE
      WRITE(6,322) TDIST,LDDEV,LNDEV
      CALL ELAPSE(ITIME)
      TIME=TIME+FLOAT(ITIME)/1000.
      WRITE(6,323) TIME
      WRITE(6,450)
  450 FORMAT(1H ,////////////,' HIT <RTN> TO CONTINUE')
      CALL TINPUT(MMMM)
C
C
C     STORE CURRENT SOLUTION
C
C
      STDLIM(SOLNO)=DLIMIT
      STLDVL(SOLNO)=LDDVLM
      STLNVL(SOLNO)=LNDVLM
      STMNLD(SOLNO)=MINLD
      STMXLD(SOLNO)=MAXLD
      STMNLN(SOLNO)=MINLN
      STMXLN(SOLNO)=MAXLN
```

```
      STNT(SOLNO)=NT
      STOBJ(SOLNO)=OBJ
      DO 130 J=1,IROUTE
        STDEP(SOLNO,J)=DEPOT(J)
        STLNTH(SOLNO,J)=LENGTH(J)
  130   STLOAD(SOLNO,J)=LOAD(J)
      DO 140 J=1,NCITY+IROUTE
        STCULD(SOLNO,J)=CUMLD(J)
        STCULN(SOLNO,J)=CUMLN(J)
        STLOKK(SOLNO,J)=LOKK(J)
        STPRED(SOLNO,J)=PRED(J)
        STSUCC(SOLNO,J)=SUCC(J)
  140   STTRK(SOLNO,J)=TRUCK(J)
      DO 145 I=1,7
      DO 145 J=1,NT
  145    STITRD(SOLNO,I,J)=ITRADE(I,J)
C
C
      GOTO 150
C
C
C
C
C     DISPLAY MENU, PROBLEM STATUS, AND TRADEOFF INFORMATION
C
C
C
  150 CONTINUE
      CALL NEWPAG
      CALL ANMODE
      CALL HOME
      WRITE(6,420) PNAME,SOLNO
      LARGE=999999
      IF(OBJ.EQ.'TOTAL DISTANCE') WRITE(6,406) TDIST,LARGE,LOAD(1),
     *   LENGTH(1),LDDEV,LIMIT1,LOAD(2),LENGTH(2),LNDEV,LIMIT2,LOAD(3),
     *   LENGTH(3)
      IF(OBJ.EQ.'LENGTH DEVIATION') WRITE(6,406) TDIST,LIMIT1,LOAD(1),
     *   LENGTH(1),LDDEV,LIMIT2,LOAD(2),LENGTH(2),LNDEV,LARGE,LOAD(3),
     *   LENGTH(3)
      IF(OBJ.EQ.'LOAD DEVIATION') WRITE(6,406) TDIST,LIMIT1,LOAD(1),
     *   LENGTH(1),LDDEV,LARGE,LOAD(2),LENGTH(2),LNDEV,LIMIT2,LOAD(3),
     *   LENGTH(3)
      IF(IROUTE.GE.4) WRITE(6,407) LOAD(4),LENGTH(4)
      IF(IROUTE.LT.4) WRITE(6,408)
      IF(IROUTE.GE.5) WRITE(6,409) LOAD(5),LENGTH(5)
      IF(IROUTE.LT.5) WRITE(6,410)
      IF(IROUTE.GE.6) WRITE(6,431) LOAD(6),LENGTH(6)
      IF(IROUTE.LT.6) WRITE(6,432)
      IF(IROUTE.GE.7) WRITE(6,435) LOAD(7),LENGTH(7)
      IF(IROUTE.LT.7) WRITE(6,436)
      IF(IROUTE.GE.8) WRITE(6,433) LOAD(8),LENGTH(8)
      IF(IROUTE.LT.8) WRITE(6,434)
      IF(IROUTE.GE.9) WRITE(6,411) ((I,LOAD(I),LENGTH(I)),I=9,IROUTE)
      WRITE(6,437) (I,I=1,MINO(6,NT))
      WRITE(6,438) OBJ,(-ITRADE(1,I),I=1,MINO(6,NT))
      WRITE(6,439) CNSTR1,(ITRADE(2,I),I=1,MINO(6,NT))
      WRITE(6,439) CNSTR2,(ITRADE(3,I),I=1,MINO(6,NT))
      WRITE(6,455) NTRADE,NT
  455 FORMAT(1H ,T12,'ORIGINAL TRADEOFFS',I4,'     REDUCED TRADEOFFS',I4)
      WRITE(6,412)
C
C
C
C     READ MENU OPTION
C
C
      READ(6,*) MENU
      IF(MENU.EQ.8) THEN
        CALL FINITT(0,700)
        STOP
      ENDIF
```

```
C
C
C
C     MENU OPTION 1:  MINIMIZE TOTAL DISTANCE
C
C
      IF(MENU.EQ.1) THEN
        OBJ='TOTAL DISTANCE'
        CNSTR1='LOAD DEVIATION'
        CNSTR2='LENGTH DEVIATION'
        IF(OLDOBJ.NE.1) THEN
          OLDOBJ=1
          GOTO 160
        ENDIF
        OLDOBJ=1
        WRITE(6,440)
        READ(5,*) NUM
        IF(NUM.EQ.0) GOTO 160
C       ELSE
        LDDVLM=LDDEV+MAXO(0,ITRADE(2,NUM))
        LNDVLM=LNDEV+MAXO(0,ITRADE(3,NUM))
        LIMIT1=LDDVLM
        LIMIT2=LNDVLM
        IF(ITRADE(6,NUM).EQ.0) CALL XCHNG2(ITRADE(4,NUM),ITRADE(5,NUM))
        IF(ITRADE(6,NUM).NE.0) CALL XCHNG3(ITRADE(4,NUM),ITRADE(5,NUM),
     *                                     ITRADE(6,NUM),ITRADE(7,NUM))
        IMAXLN=-99
        IMAXLD=-99
        IMINLN=999999
        IMINLD=999999
        DO 151 K=1,IROUTE
          IF(LOKK(DEPOT(K)).NE.0) GOTO 151
          IF(LENGTH(K).GT.IMAXLN) IMAXLN=LENGTH(K)
          IF(LENGTH(K).LT.IMINLN.AND.LENGTH(K).GT.0) IMINLN=LENGTH(K)
          IF(LOAD(K).GT.IMAXLD) IMAXLD=LOAD(K)
          IF(LOAD(K).LT.IMINLD.AND.LOAD(K).GT.0) IMINLD=LOAD(K)
  151   CONTINUE
        MAXLN=IMAXLN
        MAXLD=IMAXLD
        MINLN=IMINLN
        MINLD=IMINLD
        GOTO 161
  160   WRITE(6,414) OBJ
        WRITE(6,415) CNSTR1
        READ(5,*) LDDVLM
        LIMIT1=LDDVLM
        WRITE(6,415) CNSTR2
        READ(5,*) LNDVLM
        LIMIT2=LNDVLM
        CALL NEWPAG
  161   WRITE(6,404) OBJ,CNSTR1,LIMIT1,CNSTR2,LIMIT2
        DO 2000 IRUN=1,3
          IF(IRUN.EQ.1) CALL TWOOPT(0)
          TDIST=0
          DO 180 I=1,IROUTE
  180        TDIST=TDIST+LENGTH(IROUTE)
          DSTRLX=FLOAT(TDIST)*0.1*FLOAT(IRUN-1)
          CALL THROPT(DSTRLX)
          TDIST=0
          DO 181 I=1,IROUTE
  181       TDIST=TDIST+LENGTH(I)
          IF(IRUN.EQ.1.OR.TDIST.LT.BESTDS) THEN
            BNTRAD=NTRADE
            DO 190 I=1,NTRADE
            DO 190 J=1,7
  190         BTRADE(J,I)=TRADE(J,I)
            TDIST=0
            DO 200 I=1,IROUTE
              TDIST=TDIST+LENGTH(I)
              BESTLD(I)=LOAD(I)
              BESTLN(I)=LENGTH(I)
```

```
 200        BDEPOT(I)=DEPOT(I)
          BESTDS=TDIST
          BMAXLD=MAXLD
          BMINLD=MINLD
          BMINLN=MINLN
          BMAXLN=MAXLN
          DO 201 I=1,NCITY+IROUTE
            BESTP(I)=PRED(I)
            BESTS(I)=SUCC(I)
            BESTTR(I)=TRUCK(I)
            BCUMLN(I)=CUMLN(I)
 201        BCUMLD(I)=CUMLD(I)
        ENDIF
2000  CONTINUE
        TDIST=BESTDS
        MAXLD=BMAXLD
        MINLD=BMINLD
        MAXLN=BMAXLN
        MINLN=BMINLN
        DO 203 I=1,IROUTE
          LOAD(I)=BESTLD(I)
          LENGTH(I)=BESTLN(I)
 203      DEPOT(I)=BDEPOT(I)
        DO 204 I=1,NCITY+IROUTE
          PRED(I)=BESTP(I)
          SUCC(I)=BESTS(I)
          TRUCK(I)=BESTTR(I)
          CUMLN(I)=BCUMLN(I)
 204      CUMLD(I)=BCUMLD(I)
        NTRADE=BNTRAD
        DO 205 I=1,NTRADE
        DO 205 J=1,7
 205      TRADE(J,I)=BTRADE(J,I)
        IF(NTRADE.GT.O) CALL VSORA(TRADE,7,7,NTRADE,1,WK,IER)
        NT=O
        IF(NTRADE.EQ.O) GOTO 93
C       ELSE
        PREV1=999999.
        PREV2=-999999.
        PREV3=-999999.
        DO 207 I=1,NTRADE
          IF(TRADE(1,I).NE.PREV1) THEN
            NT=NT+1
            ITRADE(1,NT)=TRADE(1,I)
            ITRADE(2,NT)=TRADE(2,I)
            ITRADE(3,NT)=TRADE(3,I)
            ITRADE(4,NT)=TRADE(4,I)
            ITRADE(5,NT)=TRADE(5,I)
            ITRADE(6,NT)=TRADE(6,I)
            ITRADE(7,NT)=TRADE(7,I)
            PREV1=TRADE(1,I)
            PREV2=TRADE(2,I)
            PREV3=TRADE(3,I)
            GOTO 207
          ENDIF
          IF(TRADE(2,I).LE.PREV2.AND.TRADE(3,I).LE.PREV3) THEN
            ITRADE(2,NT)=TRADE(2,I)
            ITRADE(3,NT)=TRADE(3,I)
            ITRADE(4,NT)=TRADE(4,I)
            ITRADE(5,NT)=TRADE(5,I)
            ITRADE(6,NT)=TRADE(6,I)
            ITRADE(7,NT)=TRADE(7,I)
            PREV1=TRADE(1,I)
            PREV2=TRADE(2,I)
            PREV3=TRADE(3,I)
          ENDIF
 207    CONTINUE
        CALL NONDOM(ITRADE,NT)
      GOTO 93
    ENDIF
C
```

```
C
C
C     MENU OPTION 2:  MINIMIZE ROUTE-LOAD DEVIATION
C
C
      IF(MENU.EQ.2) THEN
        OBJ='LOAD DEVIATION'
        CNSTR1='TOTAL DISTANCE'
        CNSTR2='LENGTH DEVIATION'
        IF(OLDOBJ.NE.2) THEN
          OLDOBJ=2
          GOTO 208
        ENDIF
        OLDOBJ=2
        WRITE(6,440)
        READ(5,*) NUM
        IF(NUM.EQ.0) GOTO 208
C       ELSE
        DLIMIT=TDIST+MAXO(0,ITRADE(2,NUM))
        LNDVLM=LNDEV+MAXO(0,ITRADE(3,NUM))
        LIMIT1=DLIMIT
        LIMIT2=LNDVLM
        IP1=ITRADE(4,NUM)
        IP2=ITRADE(5,NUM)
        IP3=ITRADE(6,NUM)
        ITYPE=ITRADE(7,NUM)
        IF(ITYPE.LE.4) CALL XCHNG3(IP1,IP2,IP3,ITYPE)
        IF(ITYPE.EQ.5) CALL XCHNG2(IP1,IP2)
        IF(ITYPE.EQ.6) CALL XCHNG2(IP2,IP3)
        IF(ITYPE.EQ.7) CALL XCHNG2(IP1,IP3)
        DO 2002 IRT=1,IROUTE
          START=DEPOT(IRT)
          IF(SUCC(START).GT.NCITY.OR.SUCC(SUCC(START)).GT.NCITY)
     *                                                     GOTO 2002
          NEXT=SUCC(START)
 2001     NODE=NEXT
          IF(SUCC(NODE).GT.NCITY) THEN
            END=NODE
            CALL TSP(START,END,PRED,SUCC,LNGTH,DIST,ALLOW)
            LENGTH(IRT)=LNGTH
            GOTO 2002
          ENDIF
          NEXT=SUCC(NODE)
          GOTO 2001
 2002   CONTINUE
        NEXT=NCITY+1
        INSIDE=0
 2003   NODE=NEXT
        IF(NODE.EQ.NCITY+1.AND.INSIDE.EQ.1) GOTO 2004
        INSIDE=1
        IF(NODE.GT.NCITY) THEN
          ITRK=TRUCK(NODE)
          ILD=0
          ILN=0
        ENDIF
        ILD=ILD+DEMAND(NODE)
        IF(NODE.LE.NCITY) ILN=ILN+DIST(NODE,PRED(NODE)) + ALLOW
        CUMLD(NODE)=ILD
        CUMLN(NODE)=ILN
        TRUCK(NODE)=ITRK
        FEASLD(ITRK)=ILD
        LOAD(ITRK)=ILD
        NEXT=SUCC(NODE)
        IF(NEXT.GT.NCITY) FEASLN(ITRK)=ILN+DIST(NODE,NEXT)
        IF(NEXT.GT.NCITY) LENGTH(ITRK)=ILN+DIST(NODE,NEXT)
        GOTO 2003
 2004   TDIST=0
        DO 2005 I=1,IROUTE
 2005   TDIST=TDIST+LENGTH(I)
        IMAXLN=-99
        IMAXLD=-99
```

```
      IMINLN=999999
      IMINLD=999999
      DO 206  K=1,IROUTE
        IF(LOKK(DEPOT(K)).NE.O) GOTO 206
        IF(LENGTH(K).GT.IMAXLN) IMAXLN=LENGTH(K)
        IF(LENGTH(K).LT.IMINLN.AND.LENGTH(K).GT.O) IMINLN=LENGTH(K)
        IF(LOAD(K).GT.IMAXLD) IMAXLD=LOAD(K)
        IF(LOAD(K).LT.IMINLD.AND.LOAD(K).GT.O) IMINLD=LOAD(K)
206   CONTINUE
      MAXLN=IMAXLN
      MAXLD=IMAXLD
      MINLN=IMINLN
      MINLD=IMINLD
      GOTO 209
208   WRITE(6,414) OBJ
      WRITE(6,415) CNSTR1
      READ(5,*) DLIMIT
      LIMIT1=DLIMIT
      WRITE(6,415) CNSTR2
      READ(5,*) LNDVLM
      LIMIT2=LNDVLM
      CALL NEWPAG
209   WRITE(6,404) OBJ,CNSTR1,LIMIT1,CNSTR2,LIMIT2
      NT=O
      DO 213 IRUN=1,3
        IF(IRUN.EQ.1) CALL LDDV2(O)
        LDDEV=MAXLD-MINLD
        LDRLX=FLOAT(LDDEV)*.5*FLOAT(IRUN-1)
        CALL LDDV3(LDRLX)
        LDDEV=MAXLD-MINLD
        IF(IRUN.EQ.1.OR.LDDEV.LT.BLDDEV) THEN
          DO 210 I=1,IROUTE
            BESTLD(I)=LOAD(I)
            BESTLN(I)=LENGTH(I)
210         BDEPOT(I)=DEPOT(I)
          DO 211 I=1,NCITY+IROUTE
            BESTP(I)=PRED(I)
            BESTS(I)=SUCC(I)
            BESTTR(I)=TRUCK(I)
            BCUMLN(I)=CUMLN(I)
211         BCUMLD(I)=CUMLD(I)
          BNTRAD=NTRADE
          DO 212 I=1,NTRADE
          DO 212 J=1,7
212         BTRADE(J,I)=TRADE(J,I)
          BMAXLD=MAXLD
          BMINLD=MINLD
          BMAXLN=MAXLN
          BMINLN=MINLN
          LDDEV=MAXLD-MINLD
          BLDDEV=LDDEV
        ENDIF
213   CONTINUE
        LDDEV=BLDDEV
        MAXLD=BMAXLD
        MINLD=BMINLD
        MAXLN=BMAXLN
        MINLN=BMINLN
        DO 214 I=1,IROUTE
          LOAD(I)=BESTLD(I)
          LENGTH(I)=BESTLN(I)
214       DEPOT(I)=BDEPOT(I)
        DO 215 I=1,NCITY+IROUTE
          PRED(I)=BESTP(I)
          SUCC(I)=BESTS(I)
          TRUCK(I)=BESTTR(I)
          CUMLN(I)=BCUMLN(I)
215       CUMLD(I)=BCUMLD(I)
        NTRADE=BNTRAD
        DO 216 I=1,NTRADE
        DO 216 J=1,7
```

```
  216         TRADE(J,I)=BTRADE(J,I)
        IF(NTRADE.GT.O) CALL VSORA(TRADE,7,7,NTRADE,1,WK,IER)
        NT=O
        IF(NTRADE.EQ.O) GOTO 93
        PREV1=999999.
        PREV2=-999999.
        PREV3=-999999.
        DO 218 I=1,NTRADE
          IF(TRADE(1,I).NE.PREV1) THEN
            NT=NT+1
            ITRADE(1,NT)=TRADE(1,I)
            ITRADE(2,NT)=TRADE(2,I)
            ITRADE(3,NT)=TRADE(3,I)
            ITRADE(4,NT)=TRADE(4,I)
            ITRADE(5,NT)=TRADE(5,I)
            ITRADE(6,NT)=TRADE(6,I)
            ITRADE(7,NT)=TRADE(7,I)
            PREV1=TRADE(1,I)
            PREV2=TRADE(2,I)
            PREV3=TRADE(3,I)
            GOTO 218
          ENDIF
          IF(TRADE(2,I).LE.PREV2.AND.TRADE(3,I).LE.PREV3) THEN
            ITRADE(2,NT)=TRADE(2,I)
            ITRADE(3,NT)=TRADE(3,I)
            ITRADE(4,NT)=TRADE(4,I)
            ITRADE(5,NT)=TRADE(5,I)
            ITRADE(6,NT)=TRADE(6,I)
            ITRADE(7,NT)=TRADE(7,I)
            PREV1=TRADE(1,I)
            PREV2=TRADE(2,I)
            PREV3=TRADE(3,I)
          ENDIF
  218   CONTINUE
        CALL NONDOM(ITRADE,NT)
        GOTO 93
      ENDIF
C
C
C
C     MENU OPTION 3:  MINIMIZE ROUTE-LENGTH DEVIATION
C
C
      IF(MENU.EQ.3) THEN
        OBJ='LENGTH DEVIATION'
        CNSTR1='TOTAL DISTANCE'
        CNSTR2='LOAD DEVIATION'
        IF(OLDOBJ.NE.3) THEN
          OLDOBJ=3
          GOTO 220
        ENDIF
        OLDOBJ=3
        WRITE(6,440)
        READ(5,*) NUM
        IF(NUM.EQ.O) GOTO 220
C       ELSE
        DLIMIT=TDIST+MAXO(O,ITRADE(2,NUM))
        LDDVLM=LDDEV+MAXO(O,ITRADE(3,NUM))
        LIMIT1=DLIMIT
        LIMIT2=LDDVLM
        IP1=ITRADE(4,NUM)
        IP2=ITRADE(5,NUM)
        IP3=ITRADE(6,NUM)
        ITYPE=ITRADE(7,NUM)
        IF(ITYPE.LE.4) CALL XCHNG3(IP1,IP2,IP3,ITYPE)
        IF(ITYPE.EQ.5) CALL XCHNG2(IP1,IP2)
        IF(ITYPE.EQ.6) CALL XCHNG2(IP2,IP3)
        IF(ITYPE.EQ.7) CALL XCHNG2(IP1,IP3)
        DO 3002 IRT=1,IROUTE
          START=DEPOT(IRT)
          IF(SUCC(START).GT.NCITY.OR.SUCC(SUCC(START)).GT.NCITY)
```

```
     *                                                  GOTO 3002
          NEXT=SUCC(START)
 3001     NODE=NEXT
          IF(SUCC(NODE).GT.NCITY) THEN
            END=NODE
            CALL TSP(START,END,PRED,SUCC,LNGTH,DIST,ALLOW)
            LENGTH(IRT)=LNGTH
            GOTO 3002
          ENDIF
          NEXT=SUCC(NODE)
          GOTO 3001
 3002   CONTINUE
        NEXT=NCITY+1
        INSIDE=0
 3003   NODE=NEXT
        IF(NODE.EQ.NCITY+1.AND.INSIDE.EQ.1) GOTO 3004
        INSIDE=1
        IF(NODE.GT.NCITY) THEN
          ITRK=TRUCK(NODE)
          ILD=0
          ILN=0
        ENDIF
        ILD=ILD+DEMAND(NODE)
        IF(NODE.LE.NCITY) ILN=ILN+DIST(NODE,PRED(NODE)) + ALLOW
        CUMLD(NODE)=ILD
        CUMLN(NODE)=ILN
        TRUCK(NODE)=ITRK
        FEASLD(ITRK)=ILD
        LOAD(ITRK)=ILD
        NEXT=SUCC(NODE)
        IF(NEXT.GT.NCITY) FEASLN(ITRK)=ILN+DIST(NODE,NEXT)
        IF(NEXT.GT.NCITY) LENGTH(ITRK)=ILN+DIST(NODE,NEXT)
        GOTO 3003
 3004   TDIST=0
        DO 3005 I=1,IROUTE
 3005   TDIST=TDIST+LENGTH(I)
        IMAXLN=-99
        IMAXLD=-99
        IMINLN=999999
        IMINLD=999999
        DO 219 K=1,IROUTE
          IF(LOKK(DEPOT(K)).NE.0) GOTO 219
          IF(LENGTH(K).GT.IMAXLN) IMAXLN=LENGTH(K)
          IF(LENGTH(K).LT.IMINLN.AND.LENGTH(K).GT.0) IMINLN=LENGTH(K)
          IF(LOAD(K).GT.IMAXLD) IMAXLD=LOAD(K)
          IF(LOAD(K).LT.IMINLD.AND.LOAD(K).GT.0) IMINLD=LOAD(K)
  219   CONTINUE
        MAXLN=IMAXLN
        MAXLD=IMAXLD
        MINLN=IMINLN
        MINLD=IMINLD
        GOTO 221
  220   WRITE(6,414) OBJ
        WRITE(6,415) CNSTR1
        READ(5,*) DLIMIT
        LIMIT1=DLIMIT
        WRITE(6,415) CNSTR2
        READ(5,*) LODVLM
        LIMIT2=LODVLM
        CALL NEWPAG
  221   WRITE(6,404) OBJ,CNSTR1,LIMIT1,CNSTR2,LIMIT2
        NT=0
        CALL LNDV2(0)
        CALL LNDV3(0)
        BNTRAD=NTRADE
        DO 225 I=1,NTRADE
        DO 225 J=1,7
  225     BTRADE(J,I)=TRADE(J,I)
        DO 227 I=1,IROUTE
          BESTLD(I)=LOAD(I)
          BESTLN(I)=LENGTH(I)
```

```
227     BDEPOT(I)=DEPOT(I)
      DO 228 I=1,NCITY+IROUTE
        BESTP(I)=PRED(I)
        BESTS(I)=SUCC(I)
        BESTTR(I)=TRUCK(I)
        BCUMLN(I)=CUMLN(I)
228     BCUMLD(I)=CUMLD(I)
      LNDEV=MAXLN-MINLN
      BMAXLD=MAXLD
      BMINLD=MINLD
      BMAXLN=MAXLN
      BMINLN=MINLN
      BLNDEV=LNDEV
      LNRLX=FLOAT(LNDEV)*0.5
      CALL LNDV3(LNRLX)
      LNDEV=MAXLN-MINLN
      IF(LNDEV.GE.BLNDEV) THEN
        LNDEV=BLNDEV
        MAXLD=BMAXLD
        MINLD=BMINLD
        MAXLN=BMAXLN
        MINLN=BMINLN
        DO 229 I=1,IROUTE
          LOAD(I)=BESTLD(I)
          LENGTH(I)=BESTLN(I)
229       DEPOT(I)=BDEPOT(I)
        DO 230 I=1,NCITY+IROUTE
          PRED(I)=BESTP(I)
          SUCC(I)=BESTS(I)
          TRUCK(I)=BESTTR(I)
          CUMLN(I)=BCUMLN(I)
230       CUMLD(I)=BCUMLD(I)
        NTRADE=BNTRAD
        DO 231 I=1,NTRADE
        DO 231 J=1,7
231       TRADE(J,I)=BTRADE(J,I)
      ENDIF
      IF(NTRADE.GT.0) CALL VSORA(TRADE,7,7,NTRADE,1,WK,IER)
      NT=0
      IF(NTRADE.EQ.0) GOTO 93
      PREV1=999999.
      PREV2=-999999.
      PREV3=-999999.
      DO 233 I=1,NTRADE
        IF(TRADE(1,I).NE.PREV1) THEN
          NT=NT+1
          ITRADE(1,NT)=TRADE(1,I)
          ITRADE(2,NT)=TRADE(2,I)
          ITRADE(3,NT)=TRADE(3,I)
          ITRADE(4,NT)=TRADE(4,I)
          ITRADE(5,NT)=TRADE(5,I)
          ITRADE(6,NT)=TRADE(6,I)
          ITRADE(7,NT)=TRADE(7,I)
          PREV1=TRADE(1,I)
          PREV2=TRADE(2,I)
          PREV3=TRADE(3,I)
          GOTO 233
        ENDIF
        IF(TRADE(2,I).LE.PREV2.AND.TRADE(3,I).LE.PREV3) THEN
          ITRADE(2,NT)=TRADE(2,I)
          ITRADE(3,NT)=TRADE(3,I)
          ITRADE(4,NT)=TRADE(4,I)
          ITRADE(5,NT)=TRADE(5,I)
          ITRADE(6,NT)=TRADE(6,I)
          ITRADE(7,NT)=TRADE(7,I)
          PREV1=TRADE(1,I)
          PREV2=TRADE(2,I)
          PREV3=TRADE(3,I)
        ENDIF
233   CONTINUE
      CALL NONDOM(ITRADE,NT)
```

```
          GOTO 93
        ENDIF
C
C
C
C       MENU OPTION 4:  MANUAL ROUTE ADJUSTMENT
C
C
        IF(MENU.EQ.4) THEN
          WRITE(6,442)
          READ(5,*) I
          CALL ADJUST(I)
          LNDEV=MAXLN-MINLN
          GOTO 150
        ENDIF
C
C
C
C       MENU OPTION 5:  DISPLAY A PRIOR SOLUTION
C
C
        IF(MENU.EQ.5) THEN
          WRITE(6,441)
          READ(5,*) I
          CALL DISPLA(I,IROUTE,NCITY,PNAME,XCOORD,YCOORD,TIME)
          GOTO 150
        ENDIF
C
C
C       MENU OPTION 6:  BACKTRACK TO A PRIOR SOLUTION
C
C
        IF(MENU.EQ.6) THEN
          WRITE(6,441)
          READ(5,*) NUMBER
          CALL BKTRAK(NUMBER,ITRADE,NT,OBJ,OLDOBJ,LIMIT1,LIMIT2,CNSTR1,
     *                CNSTR2)
          SOLNO=NUMBER-1
          GOTO 93
        ENDIF
C
C
C       MENU OPTION 7:  REMOVE A ROUTE FROM CALCULATIONS.
C
C
        IF(MENU.EQ.7) THEN
          WRITE(6,442)
          READ(5,*) I
          CALL LOCK  (I,LOKK,NCITY,TRUCK,IROUTE)
          MAXLN=-999999
          MAXLD=-999999
          MINLN=999999
          MINLD=999999
          DO 250 I=1,IROUTE
            IF(LOKK(DEPOT(I)).EQ.1) GOTO 250
            IF(LOAD(I).GT.MAXLD) MAXLD=LOAD(I)
            IF(LENGTH(I).GT.MAXLN) MAXLN=LENGTH(I)
            IF(LOAD(I).LT.MINLD.AND.LOAD(I).NE.0) MINLD=LOAD(I)
            IF(LENGTH(I).LT.MINLN.AND.LENGTH(I).NE.0) MINLN=LENGTH(I)
  250     CONTINUE
          LDDEV=MAXLD-MINLD
          LNDEV=MAXLN-MINLN
            DO 260 I=1,NT
              IP1=ITRADE(4,I)
              IP2=ITRADE(5,I)
              IP3=ITRADE(6,I)
              IF(LOKK(IP1).EQ.1.OR.LOKK(IP2).EQ.1.OR.LOKK(IP3).EQ.1) THEN
                ITRADE(1,I)=9999999
                ITRADE(2,I)=9999999
                ITRADE(3,I)=9999999
              ENDIF
```

```
260     CONTINUE
      GOTO 150
    ENDIF
100 FORMAT(A)
101 FORMAT(2I3,I5,I5,I3)
102 FORMAT(3(I3,1X),I4)
103 FORMAT(2I3,I4)
255 FORMAT(1H ,T4,I3)
301 FORMAT(1H ,T11,I3,T20,I3,T33,I3,T42,I4,T50,I2,T57,I2,T63,I4,
   *2I9,2I4)
400 FORMAT(1H ,/////' FINAL SOLUTION?'/' (Y/N) ')
402 FORMAT(1H ,'OBJECTIVE TO MINIMIZE'
   */,T2,'1. LENGTH DEVIATION'/T2,'2. LOAD DEVIATION')
404 FORMAT(1H ,A,' PROBLEM BEING SOLVED SUBJECT TO FOLLOWING LIMITS:'/
   *T4,A,I5,/T4,A,I5)
405 FORMAT(1H //'  SOLUTION NUMBER',I2//3X,A/' MINIMIZATION PROBLEM'//
   *' ROUTE  LOAD  LENGTH'/)
406 FORMAT(1H ,T19,'MAIN MENU',T43,' STATUS  LIMIT',T66,'ROUTES:
   *'/T62,'#  LOAD DIST'/
   *T10,'1. MINIMIZE TOTAL DISTANCE      --',I6,3X,I4,T62,'1',I5,I6/
   *T10,'2. MINIMIZE LOAD DEVIATION      --',I6,3X,I4,T62,'2',I5,I6/
   *T10,'3. MINIMIZE LENGTH DEVIATION    --',I6,3X,I4,T62,'3',I5,I6)
407 FORMAT(1H ,T10,'4. MANUAL ROUTE IMPROVEMENT',T62,'4',I5,I6)
408 FORMAT(1H ,T10,'4. MANUAL ROUTE IMPROVEMENT')
409 FORMAT(1H ,T10,'5. DISPLAY PREVIOUS SOLUTION',T62,'5',I5,I6)
410 FORMAT(1H ,T10,'5. DISPLAY PREVIOUS SOLUTION')
411 FORMAT(1H ,T61,I2,I5,I6)
412 FORMAT(1H ,T10,'SELECT FROM MENU')
413 FORMAT(I1)
414 FORMAT(1H ,'SPECIFY NEW LIMITS FOR ',A,' PROBLEM')
415 FORMAT(1H ,A)
420 FORMAT(1H ,T20,'WORKLOAD-BALANCED VEHICLE ROUTING PROGRAM'
   *,T20,A,/T30,'SOLUTION NUMBER',I3,/)
430 FORMAT(1H ,T10,'HARDCOPY WANTED? (Y/N)')
431 FORMAT(1H ,T10,'6. BACKTRACK TO PREV. SOL.',T62,'6',I5,I6)
432 FORMAT(1H ,T10,'6. BACKTRACK TO PREV. SOL.')
433 FORMAT(1H ,T10,'8. EXIT',T62,'8',I5,I6)
434 FORMAT(1H ,T10,'8. EXIT')
435 FORMAT(1H ,T10,'7. REMOVE ROUTE FROM CALC.',T62,'7',I5,I6)
436 FORMAT(1H ,T10,'7. REMOVE ROUTE FROM CALC.')
437 FORMAT(1H ,T31,'ESTIMATED TRADEOFFS:'/T40,6I5)
438 FORMAT(1H ,T12,A,1X,'IMPROVEMENT',T41,6I5)
439 FORMAT(1H ,T12,A,1X,'RELAXATION',T41,6I5)
440 FORMAT(1H ,'WHICH TRADEOFF # IS ACCEPTABLE?')
441 FORMAT(1H ,'SOLUTION NUMBER?')
442 FORMAT(1H ,'ROUTE NUMBER?')
    END
C
C
C
C
C
C
C
C
C***********************************************************************
C
C
      SUBROUTINE TWOOPT(DSTRLX)
C
C
C     THIS SUBROUTINE IMPLEMENTS THE 2-OPT DISTANCE MINIMIZATION
C     ARC EXCHANGE ALGORITHM.
C
C***********************************************************************
      CHARACTER*1 MODE
      CHARACTER*44 PNAME,IPLACE
      INTEGER EUCLID,CITY,XCOORD(0:120),YCOORD(0:120),DEMAND(0:120)
      INTEGER HEAD(120),TAIL(120),PRED(120),SUCC(120),ROUTES,TWGT,WTLIM
      INTEGER DIST,ALLOW,TDIST,DISTLM,TRUCK,IR(100),IFLAG(40),
     * PERMI(40),PERMJ(40)
```

```
      DOUBLE PRECISION DSEED
      INTEGER START,END,POINT1,POINT2,D,FEASLD(20),FEASLN(20)
      INTEGER FTRUCK,FWD,BACK,TEMTRK(120),CUMLD(120),CUMLN(120)
      INTEGER FOUND,TRCNT,TRK,TAG,D1,D2,D3,D4,D5,D6,D7,D8
      INTEGER FSTART,FEND,FPRED(120),FSUCC(120)
      INTEGER PERMPR(120),PERMSU(120),PERMTR(120)
      INTEGER DLIMIT,FEAS,DEPOT(20),DSTRLX
      DIMENSION DIST(0:120,0:120),SAVING(3,6000),SORT(6000),PERMSV(40)
      DIMENSION ISORT(6000),JSORT(6000),LOAD(120),TRUCK(120)
      DIMENSION LENGTH(120),WORK(6),LOKK(120),TRADE(7,500)
      COMMON IROUTE,NCITY,IRUN,START,END,WTLIM,DISTLM,ALLOW,NPERM,
     *ICOUNT,LDDVLM,LNDVLM,DLIMIT,MAXLD,MINLD,MAXLN,MINLN,D1,D2,D3,D4,
     *D5,D6,DEPOT,PRED,SUCC,TRUCK,DEMAND,LENGTH,LOAD,CUMLD,CUMLN,TEMTRK,
     *FEASLD,FEASLN,PERMPR,PERMSU,PERMTR,ISORT,JSORT,SORT,DIST,XCOORD,
     *YCOORD,LOKK,TRADE,NTRADE,DSEED
C
C
      DO 100 I=1,NCITY+IROUTE
        FPRED(I)=PRED(I)
  100   FSUCC(I)=SUCC(I)
      DO 101 I=1,IROUTE
        FEASLD(I)=LOAD(I)
  101   FEASLN(I)=LENGTH(I)
      START=GGUBFS(DSEED)*NCITY+1
      END=PRED(START)
      POINT1=PRED(START)
    1 POINT1=SUCC(POINT1)
      MODE='F'
      IF(POINT1.EQ.PRED(PRED(END))) RETURN
      IF(LOKK(POINT1).EQ.1) GOTO 1
      POINT2=SUCC(POINT1)
    2 POINT2=SUCC(POINT2)
      IF(POINT2.EQ.END) GOTO 1
      IF(LOKK(POINT2).EQ.1) GOTO 2
      IF(TRUCK(POINT1).NE.TRUCK(POINT2)) MODE='B'
C
C
C
C     DISTANCE REDUCTION TEST.
C
C
      IS1=SUCC(POINT1)
      IS2=SUCC(POINT2)
      D1=DIST(POINT1,IS1)
      D2=DIST(POINT2,IS2)
      D3=DIST(POINT1,POINT2)
      D4=DIST(IS1,IS2)
      IF(D1+D2+DSTRLX.LE.D3+D4) GOTO 2
      IF(DSTRLX.NE.O.AND.TRUCK(POINT1).EQ.TRUCK(POINT2)) GOTO 2
C
C
C
C
C
C
C
C
C     FEASIBILITY TEST.
C
      CALL FEAS2(POINT1,POINT2,FEAS,O,O,DSTRLX)
      IF(FEAS.EQ.O) GOTO 2
C
C
C
C
C     EXCHANGE ARCS.
C
      CALL XCHNG2(POINT1,POINT2)
C
C
C
```

```
C
C
C        ROTATE
C
C
C
   20 START=END
      END=PRED(START)
      POINT1=END
      DSTRLX=0
      GOTO 1
C
C
C
      END
C
C***********************************************************************
C
C
      SUBROUTINE THROPT(DSTRLX)
C
C
C     THIS SUBROUTINE IS USED TO IMPLEMENT A 3-OPT DISTANCE MINIMIZATION
C     ARC-EXCHANGE ALGORITHM.
C
C***********************************************************************
      CHARACTER*1 MODE12,MODE13,MODE23
      CHARACTER*44 PNAME,IPLACE
      INTEGER FMAXLD,FMINLD,FMAXLN,FMINLN
      INTEGER EUCLID,CITY,XCOORD(0:120),YCOORD(0:120),DEMAND(0:120)
      INTEGER HEAD(120),TAIL(120),PRED(120),SUCC(120),ROUTES,TWGT,WTLIM
      INTEGER DIST,ALLOW,TDIST,DISTLM,TRUCK,IR(100),IFLAG(40),
     * PERMI(40),PERMJ(40)
      DOUBLE PRECISION DSEED
      INTEGER START,END,POINT1,POINT2,D,FEASLD(20),FEASLN(20)
      INTEGER FTRUCK,FWD,BACK,TEMTRK(120),CUMLD(120),CUMLN(120)
      INTEGER FOUND,TRCNT,TRK,TAG,D1,D2,D3,D4,D5,D6,D7,D8,DSTRLX
      INTEGER FSTART,FEND,P1,P2,P3,FPRED(120),FSUCC(120)
      INTEGER PERMPR(120),PERMSU(120),PERMTR(120),DIFF
      INTEGER DLIMIT,DEPOT(20),FEAS,POINT3,TRY(120,120)
      DIMENSION DIST(0:120,0:120),SAVING(3,6000),SORT(6000),PERMSV(40)
      DIMENSION ISORT(6000),JSORT(6000),LOAD(120),TRUCK(120)
      DIMENSION LENGTH(120),WORK(6),LOKK(120)
      DIMENSION TRADE(7,500)
      COMMON IROUTE,NCITY,IRUN,START,END,WTLIM,DISTLM,ALLOW,NPERM,
     *ICOUNT,LDDVLM,LNDVLM,DLIMIT,MAXLD,MINLD,MAXLN,MINLN,D1,D2,D3,D4,
     *D5,D6,DEPOT,PRED,SUCC,TRUCK,DEMAND,LENGTH,LOAD,CUMLD,CUMLN,TEMTRK,
     *FEASLD,FEASLN,PERMPR,PERMSU,PERMTR,ISORT,JSORT,SORT,DIST,XCOORD,
     *YCOORD,LOKK,TRADE,NTRADE,DSEED
C
C
C
C
C
      DATA TRY/14400*0/
      DO 98 I=1,NCITY+IROUTE
        FPRED(I)=PRED(I)
   98   FSUCC(I)=SUCC(I)
      DO 99 I=1,IROUTE
        FEASLD(I)=LOAD(I)
   99   FEASLN(I)=LENGTH(I)
      LNDEV=MAXLN-MINLN
      LDDEV=MAXLD-MINLD
      NTRADE=0
      START=GGUBFS(DSEED)*NCITY+1
      END=PRED(START)
      POINT1=PRED(START)
    1 POINT1=SUCC(POINT1)
      IF(POINT1.EQ.PRED(PRED(END))) RETURN
      IF(LOKK(POINT1).EQ.1) GOTO 1
      IS1=SUCC(POINT1)
```

```
      D1=DIST(POINT1,IS1)
      POINT2=POINT1
      MODE12='F'
      MODE13='F'
    2 POINT2=SUCC(POINT2)
      IF(POINT2.EQ.PRED(END)) GOTO 1
      IF(LOKK(POINT2).EQ.1) GOTO 2
      IS2=SUCC(POINT2)
      D2=DIST(POINT2,IS2)
      MODE23='F'
      POINT3=POINT2
    3 POINT3=SUCC(POINT3)
      IF(TRUCK(POINT1).NE.TRUCK(POINT2)) MODE12='B'
      IF(TRUCK(POINT1).NE.TRUCK(POINT3)) MODE13='B'
      IF(TRUCK(POINT2).NE.TRUCK(POINT3)) MODE23='B'
      IF(POINT3.EQ.END) GOTO 2
      IF(LOKK(POINT3).EQ.1) GOTO 3
      IF(DSTRLX.NE.O.AND.TRUCK(POINT1).EQ.TRUCK(POINT2).AND.
     *   TRUCK(POINT1).EQ.TRUCK(POINT3)) GOTO 3
      IF(DSTRLX.GT.O) THEN
        IF(POINT1.GT.NCITY) GOTO 3
        IF(POINT2.GT.NCITY) GOTO 3
        IF(POINT3.GT.NCITY) GOTO 3
      ENDIF
      IS3=SUCC(POINT3)
      D3=DIST(POINT3,IS3)
C
C
C
C
C
C     TYPE 1 EXCHANGE DISTANCE REDUCTION TEST
C
      NTYPE=1
      D4=DIST(POINT1,POINT2)
      D5=DIST(IS1,POINT3)
      D6=DIST(IS2,IS3)
      DIFF=D1+D2+D3-D4-D5-D6
      IF(D1+D2+D3+DSTRLX.LE.D4+D5+D6) GOTO 5
C
C
C     TYPE 1 EXCHANGE FEASIBILITY TEST
C
      ASSIGN 4 TO IRTN
      GOTO 100
    4 IF(FEAS.EQ.1) GOTO 14
C
C
C
C     TYPE 2 EXCHANGE DISTANCE REDUCTION TEST
C
    5 NTYPE=2
      D4=DIST(POINT1,IS2)
      D5=DIST(POINT3,POINT2)
      D6=DIST(IS1,IS3)
      DIFF=D1+D2+D3-D4-D5-D6
      IF(D1+D2+D3+DSTRLX.LE.D4+D5+D6) GOTO 7
C
C
C     TYPE 2 EXCHANGE FEASIBILITY TEST
C
      ASSIGN 6 TO IRTN
      GOTO 100
    6 IF(FEAS.EQ.1) GOTO 14
C
C
C
C     TYPE 3 EXCHANGE DISTANCE REDUCTION TEST
C
    7 NTYPE=3
      D4=DIST(POINT1,IS2)
```

```
      D5=DIST(POINT3,IS1)
      D6=DIST(POINT2,IS3)
      DIFF=D1+D2+D3-D4-D5-D6
      IF(D1+D2+D3+DSTRLX.LE.D4+D5+D6) GOTO 9
C
C
C     TYPE 3 EXCHANGE FEASIBILITY TEST
C
      ASSIGN 8 TO IRTN
      GOTO 100
    8 IF(FEAS.EQ.1) GOTO 14
C
C
C
C     TYPE 4 EXCHANGE DISTANCE REDUCTION TEST
C
    9 NTYPE=4
      D4=DIST(POINT1,POINT3)
      D5=DIST(IS2,IS1)
      D6=DIST(POINT2,IS3)
      DIFF=D1+D2+D3-D4-D5-D6
      IF(D1+D2+D3+DSTRLX.LE.D4+D5+D6) GOTO 11
C
C
C     TYPE 4 EXCHANGE FEASIBILITY TEST
C
      ASSIGN 10 TO IRTN
      GOTO 100
   10 IF(FEAS.EQ.1) GOTO 14
C
C
C
C     TYPE 5 EXCHANGE DISTANCE REDUCTION TEST
C
   11 NTYPE=5
      D4=DIST(POINT1,POINT2)
      D5=DIST(IS1,IS2)
      DIFF=D1+D2-D4-D5
      IF(D1+D2+DSTRLX.LE.D4+D5) GOTO 12
C
C
C     TYPE 5 EXCHANGE FEASIBILITY TEST
C
      IF(TRUCK(POINT1).EQ.TRUCK(POINT2).AND.DSTRLX.NE.0) GOTO 12
      IF(TRY(POINT1,POINT2).EQ.0) THEN
        CALL FEAS2(POINT1,POINT2,FEAS,NTYPE,DIFF,DSTRLX)
        TRY(POINT1,POINT2)=1
        IF(FEAS.EQ.1) GOTO 14
      ENDIF
C
C
C
C     TYPE 6 EXCHANGE DISTANCE REDUCTION TEST
C
   12 NTYPE=6
      D4=DIST(POINT2,POINT3)
      D5=DIST(IS2,IS3)
      DIFF=D2+D3-D4-D5
      IF(D2+D3+DSTRLX.LE.D4+D5) GOTO 13
C
C
C     TYPE 6 EXCHANGE FEASIBILITY TEST
C
      IF(TRUCK(POINT2).EQ.TRUCK(POINT3).AND.DSTRLX.NE.0) GOTO 13
      IF(TRY(POINT2,POINT3).EQ.0) THEN
        CALL FEAS2(POINT2,POINT3,FEAS,NTYPE,DIFF,DSTRLX)
        TRY(POINT2,POINT3)=1
        IF(FEAS.EQ.1) GOTO 14
      ENDIF
C
C
```

```
C
C     TYPE 7 EXCHANGE DISTANCE REDUCTION TEST
C
   13 NTYPE=7
      D4=DIST(POINT1,POINT3)
      D5=DIST(IS1,IS3)
      DIFF=D1+D3-D4-D5
      IF(D1+D3+DSTRLX.LE.D4+D5) GOTO 3
C
C
C     TYPE 7 EXCHANGE FEASIBILITY TEST
C
      IF(TRUCK(POINT1).EQ.TRUCK(POINT3).AND.DSTRLX.NE.0) GOTO 3
      IF(TRY(POINT1,POINT3).EQ.1) GOTO 3
      IF(TRY(POINT1,POINT3).EQ.0) THEN
        CALL FEAS2(POINT1,POINT3,FEAS,NTYPE,DIFF,DSTRLX)
        TRY(POINT1,POINT3)=1
        IF(FEAS.EQ.0) GOTO 3
      ENDIF
C
C
C
C
C     PERFORM ARC EXCHANGE
C
   14 IF(NTYPE.LE.4) CALL XCHNG3(POINT1,POINT2,POINT3,NTYPE)
      IF(NTYPE.EQ.5) CALL XCHNG2(POINT1,POINT2)
      IF(NTYPE.EQ.6) CALL XCHNG2(POINT2,POINT3)
      IF(NTYPE.EQ.7) CALL XCHNG2(POINT1,POINT3)
C
C
C
C
C     ROTATE
C
      IRLX=DSTRLX
      DSTRLX=0
      NTRADE=0
      LDDEV=MAXLD-MINLD
      LNDEV=MAXLN-MINLN
      START=END
      IF(IRLX.GT.0) START=DEPOT(TRUCK(PRED(START)))
      END=PRED(START)
      POINT1=END
      DO 15 I=1,NCITY+IROUTE
      DO 15 J=1,NCITY+IROUTE
   15 TRY(I,J)=0
      GOTO 1
C
  100 CONTINUE
  999 FORMAT(1H ,'INSIDE FEAS3',4I5)
      FEAS=0
      P1=POINT1
      P2=POINT2
      P3=POINT3
C
      GOTO (110,120,130,140), NTYPE
C
C
C
C
C     TYPE 1 EXCHANGE
C
C
C     IF ALL 3 POINTS ARE IN THE SAME ROUTE, THE EXCHANGE IS FEASIBLE.
C
  110 IF(TRUCK(P1).EQ.TRUCK(P2).AND.TRUCK(P2).EQ.TRUCK(P3)) THEN
        FEASLN(TRUCK(P1))=LENGTH(TRUCK(P1))+D4+D5+D6-D1-D2-D3
        FEASLD(TRUCK(P1))=LOAD(TRUCK(P1))
        GOTO 150
      ENDIF
```

```
C
C
C     EACH POINT IN A DIFFERENT ROUTE:
C
      IF(TRUCK(P1).NE.TRUCK(P2).AND.TRUCK(P2).NE.TRUCK(P3).AND.TRUCK(P1)
     *  .NE.TRUCK(P3)) THEN
C
C     1ST ROUTE:
        IF(CUMLD(P1)+CUMLD(P2).GT.WTLIM) GOTO IRTN
        IF(CUMLN(P1)+CUMLN(P2)+DIST(P1,P2).GT.DISTLM) GOTO IRTN
C     2ND ROUTE:
        IF(LOAD(TRUCK(P1))-CUMLD(P1)+CUMLD(P3).GT.WTLIM) GOTO IRTN
        L1=LENGTH(TRUCK(P1))-CUMLN(IS1)+ALLOW
        IF(IS1.GT.NCITY) L1=0
        IF(L1+CUMLN(P3)+DIST(IS1,P3).GT.DISTLM) GOTO IRTN
C     3RD ROUTE:
        IF(LOAD(TRUCK(P2))-CUMLD(P2)+LOAD(TRUCK(P3))-CUMLD(P3).GT.WTLIM)
     *    GOTO IRTN
        L2=LENGTH(TRUCK(P2))-CUMLN(IS2)+ALLOW
        IF(IS2.GT.NCITY) L2=0
        L3=LENGTH(TRUCK(P3))-CUMLN(IS3)+ALLOW
        IF(IS3.GT.NCITY) L3=0
        IF(L2+L3+DIST(IS2,IS3).GT.DISTLM) GOTO IRTN
C
      FEASLD(TRUCK(P1))=CUMLD(P1)+CUMLD(P2)
      FEASLD(TRUCK(P2))=LOAD(TRUCK(P1))-CUMLD(P1)+CUMLD(P3)
      FEASLD(TRUCK(P3))=LOAD(TRUCK(P2))-CUMLD(P2)+LOAD(TRUCK(P3))
     *                  -CUMLD(P3)
      FEASLN(TRUCK(P1))=CUMLN(P1)+CUMLN(P2)+DIST(P1,P2)
      FEASLN(TRUCK(P2))=L1+CUMLN(P3)+DIST(IS1,P3)
      FEASLN(TRUCK(P3))=L2+L3+DIST(IS2,IS3)
      GOTO 150
      ENDIF
C
C
C     P1 IN ONE ROUTE; P2 & P3 IN OTHER ROUTE:
C
      IF(TRUCK(P2).EQ.TRUCK(P3).AND.TRUCK(P2).NE.TRUCK(P1)) THEN
C
C     1ST ROUTE:
        IF(CUMLD(P1)+CUMLD(P2).GT.WTLIM) GOTO IRTN
        IF(CUMLN(P1)+CUMLN(P2)+DIST(P1,P2).GT.DISTLM) GOTO IRTN
C     2ND ROUTE:
        IF(LOAD(TRUCK(P1))-CUMLD(P1)+LOAD(TRUCK(P3))-CUMLD(P2).GT.WTLIM)
     *    GOTO IRTN
        L1=LENGTH(TRUCK(P1))-CUMLN(IS1)+ALLOW
        IF(IS1.GT.NCITY) L1=0
        L2=CUMLN(P3)-CUMLN(IS2)+ALLOW
        L3=LENGTH(TRUCK(P3))-CUMLN(IS3)+ALLOW
        IF(IS3.GT.NCITY) L3=0
        IF(L1+L2+L3+DIST(IS1,P3)+DIST(IS2,IS3).GT.DISTLM) GOTO IRTN
C
      FEASLD(TRUCK(P1))=CUMLD(P1)+CUMLD(P2)
      FEASLD(TRUCK(P2))=LOAD(TRUCK(P1))-CUMLD(P1)+LOAD(TRUCK(P3))
     *                  -CUMLD(P2)
      FEASLN(TRUCK(P1))=CUMLN(P1)+CUMLN(P2)+DIST(P1,P2)
      FEASLN(TRUCK(P2))=L1+L2+L3+DIST(IS1,P3)+DIST(IS2,IS3)
      GOTO 150
      ENDIF
C
C
C     P3 IN ONE ROUTE; P1 & P2 IN OTHER ROUTE:
C
      IF(TRUCK(P1).EQ.TRUCK(P2).AND.TRUCK(P1).NE.TRUCK(P3)) THEN
C
C     IST ROUTE:
        IF(CUMLD(P2)+CUMLD(P3).GT.WTLIM) GOTO IRTN
        IF(CUMLN(P2)-DIST(P1,IS1)+CUMLN(P3)+DIST(P1,P2)+DIST(IS1,P3)
     *    .GT.DISTLM) GOTO IRTN
C     2ND ROUTE:
        IF(LOAD(TRUCK(P2))-CUMLD(P2)+LOAD(TRUCK(P3))-CUMLD(P3).GT.WTLIM)
```

```
     *    GOTO IRTN
          L1=LENGTH(TRUCK(P2))-CUMLN(IS2)+ALLOW
          IF(IS2.GT.NCITY) L1=0
          L2=LENGTH(TRUCK(P3))-CUMLN(IS3)+ALLOW
          IF(IS3.GT.NCITY) L2=0
          IF(L1+L2+DIST(IS2,IS3).GT.DISTLM) GOTO IRTN
C
        FEASLD(TRUCK(P1))=CUMLD(P2)+CUMLD(P3)
        FEASLD(TRUCK(P3))=LOAD(TRUCK(P2))-CUMLD(P2)+LOAD(TRUCK(P3))
     *                    -CUMLD(P3)
        FEASLN(TRUCK(P1))=CUMLN(P2)-DIST(P1,IS1)+CUMLN(P3)+DIST(P1,P2)
     *                    +DIST(IS1,P3)
        FEASLN(TRUCK(P3))=L1+L2+DIST(IS2,IS3)
        GOTO 150
        ENDIF
C
C
C       P2 IN ONE ROUTE; P1 & P3 IN OTHER ROUTE:
C
        IF(TRUCK(P1).EQ.TRUCK(P3).AND.TRUCK(P2).NE.TRUCK(P1)) THEN
C
C       1ST ROUTE:
          IF(LOAD(TRUCK(P2))+CUMLD(P1)-CUMLD(P3).GT.WTLIM) GOTO IRTN
          L1=CUMLN(P1)-CUMLN(IS3)+ALLOW
          L2=LENGTH(TRUCK(P2))-DIST(P2,IS2)
          IF(L1+L2+DIST(P1,P2)+DIST(IS3,IS2).GT.DISTLM) GOTO IRTN
C       2ND ROUTE:
          L3=LENGTH(TRUCK(P1))-CUMLN(IS1)+ALLOW
          IF(IS1.GT.NCITY) L3=0
          IF(L3+CUMLN(P3)+DIST(IS1,P3).GT.DISTLM) GOTO IRTN
C
        FEASLD(TRUCK(P2))=LOAD(TRUCK(P2))+CUMLD(P1)-CUMLD(P3)
        FEASLD(TRUCK(P1))=LOAD(TRUCK(P1))-CUMLD(P1)+CUMLD(P3)
        FEASLN(TRUCK(P2))=L1+L2+DIST(P1,P2)+DIST(IS3,IS2)
        FEASLN(TRUCK(P1))=L3+CUMLN(P3)+DIST(IS1,P3)
        GOTO 150
        ENDIF
C
C
C
C
C       TYPE 2 EXCHANGE
C
C
C       IF ALL 3 POINTS ARE IN THE SAME ROUTE, THE EXCHANGE IS FEASIBLE.
C
  120   IF(TRUCK(P1).EQ.TRUCK(P2).AND.TRUCK(P2).EQ.TRUCK(P3)) THEN
C
          FEASLN(TRUCK(P1))=LENGTH(TRUCK(P1))+D4+D5+D6-D1-D2-D3
          FEASLD(TRUCK(P1))=LOAD(TRUCK(P1))
          GOTO 150
        ENDIF
C
C
C       EACH POINT IN A DIFFERENT ROUTE:
C
        IF(TRUCK(P1).NE.TRUCK(P2).AND.TRUCK(P2).NE.TRUCK(P3).AND.
     *     TRUCK(P1).NE.TRUCK(P3)) THEN
C
C       1ST ROUTE:
          IF(CUMLD(P1)+LOAD(TRUCK(P2))-CUMLD(P2).GT.WTLIM) GOTO IRTN
          L1=LENGTH(TRUCK(P2))-CUMLN(IS2)+ALLOW
          IF(IS2.GT.NCITY) L1=0
          IF(L1+CUMLN(P1)+DIST(P1,IS2).GT.DISTLM) GOTO IRTN
C       2ND ROUTE:
          IF(CUMLD(P2)+CUMLD(P3).GT.WTLIM) GOTO IRTN
          IF(CUMLN(P2)+CUMLN(P3)+DIST(P2,P3).GT.DISTLM) GOTO IRTN
C       3RD ROUTE:
          IF(LOAD(TRUCK(P1))-CUMLD(P1)+LOAD(TRUCK(P3))-CUMLD(P3).GT.WTLIM)
     *    GOTO IRTN
          L2=LENGTH(TRUCK(P1))-CUMLN(IS1)+ALLOW
```

```
        IF(IS1.GT.NCITY) L2=0
        L3=LENGTH(TRUCK(P3))-CUMLN(IS3)+ALLOW
        IF(IS3.GT.NCITY) L3=0
        IF(L2+L3+DIST(IS1,IS3).GT.DISTLM) GOTO IRTN
C
      FEASLD(TRUCK(P1))=CUMLD(P1)+LOAD(TRUCK(P2))-CUMLD(P2)
      FEASLD(TRUCK(P2))=CUMLD(P2)+CUMLD(P3)
      FEASLD(TRUCK(P3))=LOAD(TRUCK(P1))-CUMLD(P1)+LOAD(TRUCK(P3))
     *                  -CUMLD(P3)
      FEASLN(TRUCK(P1))=L1+CUMLN(P1)+DIST(P1,IS2)
      FEASLN(TRUCK(P2))=CUMLN(P2)+CUMLN(P3)+DIST(P2,P3)
      FEASLN(TRUCK(P3))=L2+L3+DIST(IS1,IS3)
      GOTO 150
      ENDIF
C
C
C     P1 IN ONE ROUTE; P2 & P3 IN OTHER ROUTE:
C
      IF(TRUCK(P2).EQ.TRUCK(P3).AND.TRUCK(P1).NE.TRUCK(P2)) THEN
C
C     1ST ROUTE:
        IF(CUMLD(P1)+CUMLD(P3).GT.WTLIM) GOTO IRTN
        IF(CUMLN(P1)+CUMLN(P3)-DIST(P2,IS2)+DIST(P2,P3)+DIST(P1,IS2)
     *    .GT.DISTLM) GOTO IRTN
C     2ND ROUTE:
        IF(LOAD(TRUCK(P1))-CUMLD(P1)+LOAD(TRUCK(P2))-CUMLD(P3).GT.WTLIM)
     *    GOTO IRTN
        L1=LENGTH(TRUCK(P1))-CUMLN(IS1)+ALLOW
        IF(IS1.GT.NCITY) L1=0
        L2=LENGTH(TRUCK(P2))-CUMLN(IS3)+ALLOW
        IF(IS3.GT.NCITY) L2=0
        IF(L1+L2+DIST(IS1,IS3).GT.DISTLM) GOTO IRTN
C
      FEASLD(TRUCK(P1))=CUMLD(P1)+CUMLD(P3)
      FEASLD(TRUCK(P2))=LOAD(TRUCK(P1))-CUMLD(P1)+LOAD(TRUCK(P2))
     *                  -CUMLD(P3)
      FEASLN(TRUCK(P1))=CUMLN(P1)+CUMLN(P3)-DIST(P2,IS2)+DIST(P2,P3)
     *                  +DIST(P1,IS2)
      FEASLN(TRUCK(P2))=L1+L2+DIST(IS1,IS3)
      GOTO 150
      ENDIF
C
C
C     P3 IN ONE ROUTE; P1 & P2 IN OTHER ROUTE:
C
      IF(TRUCK(P1).EQ.TRUCK(P2).AND.TRUCK(P3).NE.TRUCK(P1)) THEN
C
C     1ST ROUTE:
        L1=LENGTH(TRUCK(P2))-CUMLN(IS2)+ALLOW
        IF(IS2.GT.NCITY) L1=0
        IF(L1+CUMLN(P1)+DIST(P1,IS2).GT.DISTLM) GOTO IRTN
C     2ND ROUTE:
        IF(LOAD(TRUCK(P3))+CUMLD(P2)-CUMLD(P1).GT.WTLIM) GOTO IRTN
        IF(LENGTH(TRUCK(P3))-DIST(P3,IS3)+CUMLN(P2)-CUMLN(IS1)+ALLOW
     *     +DIST(P2,P3)+DIST(IS1,IS3).GT.DISTLM) GOTO IRTN
C
      FEASLD(TRUCK(P1))=LOAD(TRUCK(P1))-CUMLD(P2)+CUMLD(P1)
      FEASLD(TRUCK(P3))=LOAD(TRUCK(P3))+CUMLD(P2)-CUMLD(P1)
      FEASLN(TRUCK(P1))=L1+CUMLN(P1)+DIST(P1,IS2)
      FEASLN(TRUCK(P3))=LENGTH(TRUCK(P3))-DIST(P3,IS3)+CUMLN(P2)
     *                  -CUMLN(IS1)+DIST(P2,P3)+DIST(IS1,IS3)+ALLOW
      GOTO 150
      ENDIF
C
C
C     P2 IN ONE ROUTE; P1 & P3 IN OTHER ROUTE:
C
      IF(TRUCK(P1).EQ.TRUCK(P3).AND.TRUCK(P1).NE.TRUCK(P2)) THEN
C
C     1ST ROUTE:
        IF(LOAD(TRUCK(P1))-CUMLD(P3)+LOAD(TRUCK(P2))-CUMLD(P2).GT.WTLIM)
```

```
     *    GOTO IRTN
        L1=LENGTH(TRUCK(P1))-CUMLN(IS1)+ALLOW
        IF(IS1.GT.NCITY) L1=0
        L2=LENGTH(TRUCK(P2))-CUMLN(IS2)+ALLOW
        IF(IS2.GT.NCITY) L2=0
        IF(CUMLN(P1)-DIST(P3,IS3)+L1+L2+DIST(IS1,IS3)+DIST(P1,IS2)
     *    .GT.DISTLM) GOTO IRTN
C     2ND ROUTE:
        IF(CUMLD(P3)+CUMLD(P2).GT.WTLIM) GOTO IRTN
        IF(CUMLN(P3)+CUMLN(P2)+DIST(P2,P3).GT.DISTLM) GOTO IRTN
C
      FEASLD(TRUCK(P1))=LOAD(TRUCK(P1))-CUMLD(P3)+LOAD(TRUCK(P2))
     *                  -CUMLD(P2)
      FEASLD(TRUCK(P2))=CUMLD(P3)+CUMLD(P2)
      FEASLN(TRUCK(P1))=CUMLN(P1)-DIST(P3,IS3)+L1+L2+DIST(IS1,IS3)
     *                  +DIST(P1,IS2)
      FEASLN(TRUCK(P2))=CUMLN(P3)+CUMLN(P2)+DIST(P2,P3)
      GOTO 150
      ENDIF
C
C
C
C
C     TYPE 3 EXCHANGE
C
C
C     IF ALL 3 POINTS ARE IN THE SAME ROUTE, THE EXCHANGE IS FEASIBLE.
C
  130 IF(TRUCK(P1).EQ.TRUCK(P2).AND.TRUCK(P1).EQ.TRUCK(P3)) THEN
C
        FEASLN(TRUCK(P1))=LENGTH(TRUCK(P1))+D4+D5+D6-D1-D2-D3
        FEASLD(TRUCK(P1))=LOAD(TRUCK(P1))
        GOTO 150
      ENDIF
C
C
C     EACH POINT IN A DIFFERENT ROUTE:
C
      IF(TRUCK(P1).NE.TRUCK(P2).AND.TRUCK(P2).NE.TRUCK(P3).AND.TRUCK(P1)
     *   .NE.TRUCK(P3)) THEN
C
C     1ST ROUTE:
        IF(CUMLD(P1)+LOAD(TRUCK(P2))-CUMLD(P2).GT.WTLIM) GOTO IRTN
        L1=LENGTH(TRUCK(P2))-CUMLN(IS2)+ALLOW
        IF(IS2.GT.NCITY) L1=0
        IF(CUMLN(P1)+L1+DIST(P1,IS2).GT.DISTLM) GOTO IRTN
C     2ND ROUTE:
        IF(LOAD(TRUCK(P1))-CUMLD(P1)+CUMLD(P3).GT.WTLIM) GOTO IRTN
        L2=LENGTH(TRUCK(P1))-CUMLN(IS1)+ALLOW
        IF(IS1.GT.NCITY) L2=0
        IF(L2+CUMLN(P3)+DIST(IS1,P3).GT.DISTLM) GOTO IRTN
C     3RD ROUTE:
        IF(CUMLD(P2)+LOAD(TRUCK(P3))-CUMLD(P3).GT.WTLIM) GOTO IRTN
        L3=LENGTH(TRUCK(P3))-CUMLN(IS3)+ALLOW
        IF(IS3.GT.NCITY) L3=0
        IF(CUMLN(P2)+L3+DIST(P2,IS3).GT.DISTLM) GOTO IRTN
C
      FEASLD(TRUCK(P1))=CUMLD(P1)+LOAD(TRUCK(P2))-CUMLD(P2)
      FEASLD(TRUCK(P2))=LOAD(TRUCK(P1))-CUMLD(P1)+CUMLD(P3)
      FEASLD(TRUCK(P3))=CUMLD(P2)+LOAD(TRUCK(P3))-CUMLD(P3)
      FEASLN(TRUCK(P1))=CUMLN(P1)+L1+DIST(P1,IS2)
      FEASLN(TRUCK(P2))=L2+CUMLN(P3)+DIST(IS1,P3)
      FEASLN(TRUCK(P3))=CUMLN(P2)+L3+DIST(P2,IS3)
      GOTO 150
      ENDIF
C
C
C     P1 IN ONE ROUTE; P2 & P3 IN OTHER ROUTE:
C
      IF(TRUCK(P2).EQ.TRUCK(P3).AND.TRUCK(P1).NE.TRUCK(P2)) THEN
C
```

```
C      1ST ROUTE:
         IF(LOAD(TRUCK(P1))+CUMLD(P3)-CUMLD(P2).GT.WTLIM) GOTO IRTN
         L1=CUMLN(P3)-CUMLN(IS2)+ALLOW
         IF(L1+LENGTH(TRUCK(P1))-DIST(P1,IS1)+DIST(P1,IS2)+DIST(P3,IS1)
     *      .GT.DISTLM) GOTO IRTN
C      2ND ROUTE:
         L2=LENGTH(TRUCK(P3))-CUMLN(IS3)+ALLOW
         IF(IS3.GT.NCITY) L2=0
         IF(L2+CUMLN(P2)+DIST(P2,IS3).GT.DISTLM) GOTO IRTN
C
       FEASLD(TRUCK(P1))=LOAD(TRUCK(P1))+CUMLD(P3)-CUMLD(P2)
       FEASLD(TRUCK(P2))=LOAD(TRUCK(P2))-CUMLD(P3)+CUMLD(P2)
       FEASLN(TRUCK(P1))=L1+LENGTH(TRUCK(P1))-DIST(P1,IS1)+DIST(P1,IS2)
     *                   +DIST(P3,IS1)
       FEASLN(TRUCK(P2))=L2+CUMLN(P2)+DIST(P2,IS3)
       GOTO 150
       ENDIF
C
C
C      P3 IN ONE ROUTE; P1 & P2 IN OTHER ROUTE:
C
       IF(TRUCK(P1).EQ.TRUCK(P2).AND.TRUCK(P1).NE.TRUCK(P3)) THEN
C
C      1ST ROUTE:
         L1=LENGTH(TRUCK(P2))-CUMLN(IS2)+ALLOW
         IF(IS2.GT.NCITY) L1=0
         IF(L1+CUMLN(P1)+DIST(P1,IS2).GT.DISTLM) GOTO IRTN
C      2ND ROUTE:
         IF(LOAD(TRUCK(P3))+CUMLD(P2)-CUMLD(P1).GT.WTLIM) GOTO IRTN
         L2=LENGTH(TRUCK(P3))-DIST(P3,IS3)+ALLOW
         IF(L2+CUMLN(P2)-CUMLN(IS1)+DIST(P2,IS3)+DIST(P3,IS1)+ALLOW
     *      .GT.DISTLM) GOTO IRTN
C
       FEASLD(TRUCK(P2))=LOAD(TRUCK(P2))-CUMLD(P2)+CUMLD(P1)
       FEASLD(TRUCK(P3))=LOAD(TRUCK(P3))+CUMLD(P2)-CUMLD(P1)
       FEASLN(TRUCK(P2))=L1+CUMLN(P1)+DIST(P1,IS2)
       FEASLN(TRUCK(P3))=L2+CUMLN(P2)-CUMLN(IS1)+DIST(P2,IS3)+ALLOW
     *                   +DIST(P3,IS1)
       GOTO 150
       ENDIF
C
C
C      P2 IN ONE ROUTE; P1 & P3 IN OTHER ROUTE:
C
       IF(TRUCK(P1).EQ.TRUCK(P3).AND.TRUCK(P1).NE.TRUCK(P2)) THEN
C
C      1ST ROUTE:
         IF(LOAD(TRUCK(P2))+CUMLD(P1)-CUMLD(P3).GT.WTLIM) GOTO IRTN
         L1=LENGTH(TRUCK(P2))-DIST(P2,IS2)
         IF(L1+CUMLN(P1)-CUMLN(IS3)+ALLOW+DIST(P2,IS3)+DIST(P1,IS2).GT.
     *    DISTLM) GOTO IRTN
C      2ND ROUTE:
         L2=LENGTH(TRUCK(P1))-CUMLN(IS1)+ALLOW
         IF(IS1.GT.NCITY) L2=0
         IF(L2+CUMLN(P3)+DIST(P3,IS1).GT.DISTLM) GOTO IRTN
C
       FEASLD(TRUCK(P2))=LOAD(TRUCK(P2))+CUMLD(P1)-CUMLD(P3)
       FEASLD(TRUCK(P1))=LOAD(TRUCK(P1))-CUMLD(P1)+CUMLD(P3)
       FEASLN(TRUCK(P2))=L1+CUMLN(P1)-CUMLN(IS3)+DIST(P2,IS3)+DIST(P1,IS2
     *                                                         +ALLOW)
       FEASLN(TRUCK(P1))=L2+CUMLN(P3)+DIST(P3,IS1)
       GOTO 150
       ENDIF
C
C
C
C
C      TYPE 4 EXCHANGE
C
C
C      IF ALL 3 POINTS ARE IN THE SAME ROUTE, THE EXCHANGE IS FEASIBLE.
```

```
C
  140 IF(TRUCK(P1).EQ.TRUCK(P2).AND.TRUCK(P1).EQ.TRUCK(P3)) THEN
C
        FEASLN(TRUCK(P1))=LENGTH(TRUCK(P1))+D4+D5+D6-D1-D2-D3
        FEASLD(TRUCK(P1))=LOAD(TRUCK(P1))
        GOTO 150
      ENDIF
C
C
C     EACH POINT IN A DIFFERENT ROUTE:
C
      IF(TRUCK(P1).NE.TRUCK(P2).AND.TRUCK(P2).NE.TRUCK(P3).AND.TRUCK(P1)
     *    .NE.TRUCK(P3)) THEN
C
C     1ST ROUTE:
        IF(CUMLD(P1)+CUMLD(P3).GT.WTLIM) GOTO IRTN
        IF(CUMLN(P1)+CUMLN(P3)+DIST(P1,P3).GT.DISTLM) GOTO IRTN
C     2ND ROUTE:
        IF(LOAD(TRUCK(P1))-CUMLD(P1)+LOAD(TRUCK(P2))-CUMLD(P2).GT.WTLIM)
     *     GOTO IRTN
        L1=LENGTH(TRUCK(P1))-CUMLN(IS1)+ALLOW
        IF(IS1.GT.NCITY) L1=0
        L2=LENGTH(TRUCK(P2))-CUMLN(IS2)+ALLOW
        IF(IS2.GT.NCITY) L2=0
        IF(L1+L2+DIST(IS2,IS1).GT.DISTLM) GOTO IRTN
C     3RD ROUTE:
        IF(CUMLD(P2)+LOAD(TRUCK(P3))-CUMLD(P3).GT.WTLIM) GOTO IRTN
        L3=LENGTH(TRUCK(P3))-CUMLN(IS3)+ALLOW
        IF(IS3.GT.NCITY) L3=0
        IF(CUMLN(P2)+L3+DIST(P2,IS3).GT.DISTLM) GOTO IRTN
C
      FEASLD(TRUCK(P1))=CUMLD(P1)+CUMLD(P3)
      FEASLD(TRUCK(P2))=LOAD(TRUCK(P1))-CUMLD(P1)+LOAD(TRUCK(P2))
     *                  -CUMLD(P2)
      FEASLD(TRUCK(P3))=CUMLD(P2)+LOAD(TRUCK(P3))-CUMLD(P3)
      FEASLN(TRUCK(P1))=CUMLN(P1)+CUMLN(P3)+DIST(P1,P3)
      FEASLN(TRUCK(P2))=L1+L2+DIST(IS2,IS1)
      FEASLN(TRUCK(P3))=CUMLN(P2)+L3+DIST(P2,IS3)
      GOTO 150
      ENDIF
C
C
C     P1 IN ONE ROUTE; P2 & P3 IN OTHER ROUTE:
C
      IF(TRUCK(P2).EQ.TRUCK(P3).AND.TRUCK(P1).NE.TRUCK(P2)) THEN
C
C     1ST ROUTE:
        IF(LOAD(TRUCK(P1))+CUMLD(P3)-CUMLD(P2).GT.WTLIM) GOTO IRTN
        L1=LENGTH(TRUCK(P1))-CUMLN(IS1)+ALLOW
        IF(IS1.GT.NCITY) L1=0
        IF(CUMLN(P1)+L1+CUMLN(P3)-CUMLN(IS2)+ALLOW+DIST(P1,P3)+
     *    DIST(IS2,IS1).GT.DISTLM) GOTO IRTN
C     2ND ROUTE:
        L2=LENGTH(TRUCK(P3))-CUMLN(IS3)+ALLOW
        IF(IS3.GT.NCITY) L2=0
        IF(L2+CUMLN(P2)+DIST(P2,IS3).GT.DISTLM) GOTO IRTN
C
      FEASLD(TRUCK(P1))=LOAD(TRUCK(P1))+CUMLD(P3)-CUMLD(P2)
      FEASLD(TRUCK(P2))=LOAD(TRUCK(P2))-CUMLD(P3)+CUMLD(P2)
      FEASLN(TRUCK(P1))=CUMLN(P1)+L1+CUMLN(P3)-CUMLN(IS2)+DIST(P1,P3)
     *                  +DIST(IS2,IS1)+ALLOW
      FEASLN(TRUCK(P2))=L2+CUMLN(P2)+DIST(P2,IS3)
      GOTO 150
      ENDIF
C
C
C     P3 IN ONE ROUTE; P1 & P2 IN OTHER ROUTE:
C
      IF(TRUCK(P1).EQ.TRUCK(P2).AND.TRUCK(P3).NE.TRUCK(P2)) THEN
C
C     1ST ROUTE:
```

```
        IF(CUMLD(P1)+CUMLD(P3).GT.WTLIM) GOTO IRTN
        IF(CUMLN(P1)+CUMLN(P3)+DIST(P1,P3).GT.DISTLM) GOTO IRTN
C     2ND ROUTE:
        IF(LOAD(TRUCK(P2))-CUMLD(P1)+LOAD(TRUCK(P3))-CUMLD(P3).GT.WTLIM)
     *     GOTO IRTN
        L1=LENGTH(TRUCK(P3))-CUMLN(IS3)+ALLOW
        IF(IS3.GT.NCITY) L1=0
        L2=LENGTH(TRUCK(P2))-CUMLN(IS2)+ALLOW
        IF(IS2.GT.NCITY) L2=0
        L3=CUMLN(P2)-CUMLN(IS1)+ALLOW
        IF(L1+L2+L3+DIST(IS2,IS1)+DIST(P2,IS3).GT.DISTLM) GOTO IRTN
C
      FEASLD(TRUCK(P1))=CUMLD(P1)+CUMLD(P3)
      FEASLD(TRUCK(P3))=LOAD(TRUCK(P2))-CUMLD(P1)+LOAD(TRUCK(P3))
     *                  -CUMLD(P3)
      FEASLN(TRUCK(P1))=CUMLN(P1)+CUMLN(P3)+DIST(P1,P3)
      FEASLN(TRUCK(P3))=L1+L2+L3+DIST(IS2,IS1)+DIST(P2,IS3)
      GOTO 150
      ENDIF
C
C
C     P2 IN ONE ROUTE; P1 & P3 IN OTHER ROUTE:
C
      IF(TRUCK(P1).EQ.TRUCK(P3).AND.TRUCK(P1).NE.TRUCK(P2)) THEN
C
C     1ST ROUTE:
        IF(CUMLD(P1)+CUMLD(P2).GT.WTLIM) GOTO IRTN
        L1=CUMLN(P1)-CUMLN(IS3)+ALLOW
        IF(L1+CUMLN(P3)+CUMLN(P2)+DIST(P1,P3)+DIST(P2,IS3).GT.DISTLM)
     *     GOTO IRTN
C     2ND ROUTE:
        IF(LOAD(TRUCK(P1))-CUMLD(P1)+LOAD(TRUCK(P2))-CUMLD(P2).GT.WTLIM)
     *     GOTO IRTN
        L2=LENGTH(TRUCK(P1))-CUMLN(IS1)+ALLOW
        IF(IS1.GT.NCITY) L2=0
        L3=LENGTH(TRUCK(P2))-CUMLN(IS2)+ALLOW
        IF(IS2.GT.NCITY) L3=0
        IF(L2+L3+DIST(IS2,IS1).GT.DISTLM) GOTO IRTN
C
      FEASLD(TRUCK(P1))=CUMLD(P1)+CUMLD(P2)
      FEASLD(TRUCK(P2))=LOAD(TRUCK(P1))-CUMLD(P1)+LOAD(TRUCK(P2))
     *                  -CUMLD(P2)
      FEASLN(TRUCK(P1))=L1+CUMLN(P3)+CUMLN(P2)+DIST(P1,P3)+DIST(P2,IS3)
      FEASLN(TRUCK(P2))=L2+L3+DIST(IS2,IS1)
      GOTO 150
      ENDIF
C
C
C
C
C
C
C     ROUTE LOAD DEVIATION AND ROUTE LENGTH DEVIATION TESTS:
C
C
  150 DO 151 I=1,IROUTE
      IF(I.NE.TRUCK(P1).AND.I.NE.TRUCK(P2).AND.I.NE.TRUCK(P3)) THEN
        FEASLD(I)=LOAD(I)
        FEASLN(I)=LENGTH(I)
      ENDIF
  151 CONTINUE
      FMAXLD=-99
      FMINLD=999999
      DO 155 I=1,IROUTE
        IF(LOKK(DEPOT(I)).NE.0) GOTO 155
        IF(FEASLD(I).GT.FMAXLD) FMAXLD=FEASLD(I)
        IF(FEASLD(I).LT.FMINLD.AND.FEASLD(I).NE.0) FMINLD=FEASLD(I)
  155 CONTINUE
C
      FMAXLN=-99
      FMINLN=999999
```

```
      DO 160 I=1,IROUTE
        IF(LOKK(DEPOT(I)).NE.O) GOTO 160
        IF(FEASLN(I).GT.FMAXLN) FMAXLN=FEASLN(I)
        IF(FEASLN(I).LT.FMINLN.AND.FEASLN(I).NE.O.) FMINLN=FEASLN(I)
  160 CONTINUE
C
C
C
C     TRADEOFF ANALYSIS
C
C
      IF(DSTRLX.GT.O) GOTO 161
C     ELSE
      IF(FMAXLD-FMINLD.GT.LDDVLM.OR.FMAXLN-FMINLN.GT.LNDVLM) THEN
        NTRADE=NTRADE+1
        IF(NTRADE.GT.500) NTRADE=500
        TRADE(1,NTRADE)=-DIFF
        TRADE(2,NTRADE)=FMAXLD-FMINLD-LDDEV
        TRADE(3,NTRADE)=FMAXLN-FMINLN-LNDEV
        TRADE(4,NTRADE)=P1
        TRADE(5,NTRADE)=P2
        TRADE(6,NTRADE)=P3
        TRADE(7,NTRADE)=NTYPE
      ENDIF
  161 IF(FMAXLD-FMINLD.GT.LDDVLM) GOTO IRTN
      IF(FMAXLN-FMINLN.GT.LNDVLM) GOTO IRTN
C
C
C     EXCHANGE IS FEASIBLE
C
      FEAS=1
      MAXLD=FMAXLD
      MINLD=FMINLD
      MAXLN=FMAXLN
      MINLN=FMINLN
      GOTO IRTN
      END
C
C
C
C
C*********************************************************************
C
      SUBROUTINE XCHNG2(P1,P2)
C
C
C     THIS SUBROUTINE PERFORMS ARC EXCHANGES FOR A 2-OPT ALGORITHM.
C
C*********************************************************************
C
C
      CHARACTER*1 MODE
      CHARACTER*44 PNAME,IPLACE
      INTEGER P1,P2,DEPOT1,DEPOT2,DEPOT3,DEPOT4,STACK(100),HEAD1,TAIL2
      INTEGER EUCLID,CITY,XCOORD(O:120),YCOORD(O:120),DEMAND(O:120)
      INTEGER HEAD(120),TAIL(120),PRED(120),SUCC(120),ROUTES,TWGT,WTLIM
      INTEGER DIST,ALLOW,TDIST,DISTLM,TRUCK,IR(100),IFLAG(40),
     * PERMI(40),PERMJ(40)
      DOUBLE PRECISION DSEED
      INTEGER START,END,POINT1,POINT2,D,FEASLD(20),FEASLN(20)
      INTEGER FTRUCK,FWD,BACK,TEMTRK(120),CUMLD(120),CUMLN(120)
      INTEGER FOUND,TRCNT,TRK,TAG,D1,D2,D3,D4,D5,D6,D7,D8
      INTEGER FSTART,FEND,FPRED(120),FSUCC(120)
      INTEGER PERMPR(120),PERMSU(120),PERMTR(120)
      INTEGER DLIMIT,DEPOT(20),GAP1,GAP2,TALE(20),FIRST
      DIMENSION DIST(O:120,O:120),SAVING(3,6000),SORT(6000),PERMSV(40)
      DIMENSION ISORT(6000),JSORT(6000),LOAD(120),TRUCK(120)
      DIMENSION LENGTH(120),WORK(6),LOKK(120),TRADE(7,500)
      COMMON IROUTE,NCITY,IRUN,START,END,WTLIM,DISTLM,ALLOW,NPERM,
     *ICOUNT,LDDVLM,LNDVLM,DLIMIT,MAXLD,MINLD,MAXLN,MINLN,D1,D2,D3,D4,
     *D5,D6,DEPOT,PRED,SUCC,TRUCK,DEMAND,LENGTH,LOAD,CUMLD,CUMLN,TEMTRK,
```

```
     *FEASLD,FEASLN,PERMPR,PERMSU,PERMTR,ISORT,JSORT,SORT,DIST,XCOORD,
     *YCOORD,LOKK,TRADE,NTRADE,DSEED
C
C
C
C
C      NULL EXCHANGE:
C
       NULL=1
       IF(P1.EQ.PRED(P2).OR.P1.EQ.SUCC(P2)) RETURN
       IF(P1.GT.NCITY.AND.SUCC(P2).GT.NCITY) RETURN
       IF(P2.GT.NCITY.AND.SUCC(P1).GT.NCITY) RETURN
       NULL=0
C
C
C
C      REMOVE THE ROUTES INVOLVED IN THE EXCHANGE FROM THE NETWORK AND
C      FORM A SEPARATE NETWORK WITHIN WHICH THE 2-ARC EXCHANGE WILL TAKE
C      PLACE.
C
C
       IS1=SUCC(P1)
       FIRST=DEPOT(TRUCK(P1))
       IS2=SUCC(P2)
       DO 1 I=NCITY+1,NCITY+IROUTE
     1 TAIL(DEPOT(TRUCK(PRED(I))))=PRED(I)
       NEXT=NCITY+1
       NUM=0
       INSIDE=0
     3 NODE=NEXT
       IF(NODE.EQ.NCITY+1.AND.INSIDE.EQ.1) GOTO 4
       INSIDE=1
       IF(TRUCK(NODE).EQ.TRUCK(P1).OR.TRUCK(NODE).EQ.TRUCK(P2)) THEN
         NEXT=SUCC(TAIL(NODE))
         GOTO 3
       ENDIF
       NUM=NUM+1
       HEAD(NUM)=NODE
       TALE(NUM)=TAIL(NODE)
       NEXT=SUCC(TAIL(NODE))
       GOTO 3
C
     4 IF(TRUCK(P1).EQ.TRUCK(P2)) THEN
         SUCC(TAIL(DEPOT(TRUCK(P1))))=DEPOT(TRUCK(P1))
         PRED(DEPOT(TRUCK(P1)))=TAIL(DEPOT(TRUCK(P1)))
         GOTO 5
       ENDIF
         SUCC(TAIL(DEPOT(TRUCK(P1))))=DEPOT(TRUCK(P2))
         PRED(DEPOT(TRUCK(P2)))=TAIL(DEPOT(TRUCK(P1)))
         SUCC(TAIL(DEPOT(TRUCK(P2))))=DEPOT(TRUCK(P1))
         PRED(DEPOT(TRUCK(P1)))=TAIL(DEPOT(TRUCK(P2)))
C
     5 IF(NUM.EQ.1) THEN
         SUCC(TALE(1))=HEAD(1)
         PRED(HEAD(1))=TALE(1)
         GAP1=TALE(1)
         GAP2=HEAD(1)
       ENDIF
       IF(NUM.GT.1) THEN
         DO 6 I=1,NUM-1
         SUCC(TALE(I))=HEAD(I+1)
     6   PRED(HEAD(I+1))=TALE(I)
         SUCC(TALE(NUM))=HEAD(1)
         PRED(HEAD(1))=TALE(NUM)
         GAP1=TALE(1)
         GAP2=SUCC(TALE(1))
       ENDIF
C
C
C
C      PERFORM THE EXCHANGE.
```

```
C
C
      IS1=SUCC(P1)
      IS2=SUCC(P2)
      LAST=P1
      NEXT=P2
      SUCC(P1)=P2
    7 NODE=NEXT
      IP=PRED(NODE)
      PRED(NODE)=LAST
      IF(NODE.EQ.IS2) GOTO 8
      SUCC(NODE)=IP
      IF(NODE.EQ.IS1) SUCC(NODE)=IS2
      LAST=NODE
      NEXT=SUCC(NODE)
      GOTO 7
C
C
C
C     CALCULATE ROUTE LENGTHS AND LOADS.
C
C
    8 PRED(IS2)=LAST
      NEXT=FIRST
      INSIDE=0
    9 NODE=NEXT
      IF(NODE.EQ.FIRST.AND.INSIDE.EQ.1) GOTO 10
      INSIDE=1
      IF(NODE.GT.NCITY) THEN
        ITRK=TRUCK(NODE)
        LOAD(ITRK)=0
        LENGTH(ITRK)=0
        CUMLD(NODE)=0
        CUMLN(NODE)=0
      ENDIF
      TRUCK(NODE)=ITRK
      LOAD(ITRK)=LOAD(ITRK)+DEMAND(NODE)
      LENGTH(ITRK)=LENGTH(ITRK)+DIST(NODE,SUCC(NODE))
      IF(SUCC(NODE).LE.NCITY) LENGTH(ITRK)=LENGTH(ITRK)+ALLOW
      IF(NODE.LE.NCITY) THEN
        CUMLD(NODE)=CUMLD(PRED(NODE))+DEMAND(NODE)
        CUMLN(NODE)=CUMLN(PRED(NODE))+DIST(NODE,PRED(NODE))+ALLOW
      ENDIF
      NEXT=SUCC(NODE)
      GOTO 9
C
C
C
C     RECONNECT ROUTES INVOLVED IN EXCHANGE BACK INTO ORIGINAL NETWORK.
C
C
   10 IPRED=PRED(FIRST)
      SUCC(GAP1)=FIRST
      PRED(FIRST)=GAP1
      SUCC(IPRED)=GAP2
      PRED(GAP2)=IPRED
      RETURN
      END
C
C***********************************************************************
C
      SUBROUTINE XCHNG3(P1,P2,P3,TYPE)
C
C
C     THIS SUBROUTINE PERFORMS ARC EXCHANGES FOR A 3-OPT ALGORITHM.
C
C***********************************************************************
      CHARACTER*44 PNAME,IPLACE
      INTEGER EUCLID,CITY,XCOORD(0:120),YCOORD(0:120),DEMAND(0:120)
      INTEGER HEAD(120),TAIL(120),PRED(120),SUCC(120),ROUTES,TWGT,WTLIM
      INTEGER DIST,ALLOW,TDIST,DISTLM,TRUCK,IR(100),IFLAG(40),
```

```
     * PERMI(40),PERMJ(40)
      DOUBLE PRECISION DSEED
      INTEGER START,END,POINT1,POINT2,D,FEASLD(20),FEASLN(20)
      INTEGER FTRUCK,FWD,BACK,TEMTRK(120),CUMLD(120),CUMLN(120)
      INTEGER FOUND,TRCNT,TRK,TAG,D1,D2,D3,D4,D5,D6,D7,D8,P1,P2,P3
      INTEGER FSTART,FEND,FPRED(120),FSUCC(120),GAP1,GAP2
      INTEGER PERMPR(120),PERMSU(120),PERMTR(120),LOKK(120)
      INTEGER DLIMIT,TYPE,PLINK(3),DPLINK(3),DEPOT(20),FIRST,TALE(120)
      DIMENSION DIST(0:120,0:120),SAVING(3,6000),SORT(6000),PERMSV(40)
      DIMENSION TRADE(7,500)
      DIMENSION ISORT(6000),JSORT(6000),LOAD(120),TRUCK(120)
      DIMENSION LENGTH(120),WORK(6),LINK1(3),LINK2(3)
      COMMON IROUTE,NCITY,IRUN,START,END,WTLIM,DISTLM,ALLOW,NPERM,
     *ICOUNT,LDDVLM,LNDVLM,DLIMIT,MAXLD,MINLD,MAXLN,MINLN,D1,D2,D3,D4,
     *D5,D6,DEPOT,PRED,SUCC,TRUCK,DEMAND,LENGTH,LOAD,CUMLD,CUMLN,TEMTRK,
     *FEASLD,FEASLN,PERMPR,PERMSU,PERMTR,ISORT,JSORT,SORT,DIST,XCOORD,
     *YCOORD,LOKK,TRADE,NTRADE,DSEED
C
C
C
C
C     NULL EXCHANGE:
C
      IF(P1.EQ.PRED(P2).AND.P2.EQ.PRED(P3).AND.TYPE.EQ.1) RETURN
      IF(TYPE.EQ.3.AND.P1.GT.NCITY.AND.P2.GT.NCITY.AND.P3.GT.NCITY)
     *  RETURN
C
C
C
C
C     REMOVE THE ROUTES INVOLVED IN THE EXCHANGE FROM THE NETWORK AND
C     FORM A SEPARATE NETWORK WITHIN WHICH THE 3-OPT EXCHANGE WILL TAKE
C     PLACE.
C
C
      FIRST=DEPOT(TRUCK(P1))
      IS1=SUCC(P1)
      IS2=SUCC(P2)
      IS3=SUCC(P3)
      FWD=0
      BACK=1
      DO 1 I=NCITY+1,NCITY+IROUTE
    1 TAIL(DEPOT(TRUCK(PRED(I))))=PRED(I)
      NEXT=NCITY+1
      NUM=0
      INSIDE=0
    3 NODE=NEXT
      IF(NODE.EQ.NCITY+1.AND.INSIDE.EQ.1) GOTO 4
      INSIDE=1
      IF(TRUCK(NODE).EQ.TRUCK(P1).OR.TRUCK(NODE).EQ.TRUCK(P2).OR.
     *    TRUCK(NODE).EQ.TRUCK(P3)) THEN
        NEXT=SUCC(TAIL(NODE))
        GOTO 3
      ENDIF
      NUM=NUM+1
      HEAD(NUM)=NODE
      TALE(NUM)=TAIL(NODE)
      NEXT=SUCC(TAIL(NODE))
      GOTO 3
    4 IF(TRUCK(P1).EQ.TRUCK(P2).AND.TRUCK(P1).EQ.TRUCK(P3)) THEN
        SUCC(TAIL(DEPOT(TRUCK(P1))))=DEPOT(TRUCK(P1))
        PRED(DEPOT(TRUCK(P1)))=TAIL(DEPOT(TRUCK(P1)))
        GOTO 5
      ENDIF
      IF(TRUCK(P1).NE.TRUCK(P2).AND.TRUCK(P2).NE.TRUCK(P3).AND.
     *    TRUCK(P1).NE.TRUCK(P3)) THEN
        SUCC(TAIL(DEPOT(TRUCK(P1))))=DEPOT(TRUCK(P2))
        PRED(DEPOT(TRUCK(P2)))=TAIL(DEPOT(TRUCK(P1)))
        SUCC(TAIL(DEPOT(TRUCK(P2))))=DEPOT(TRUCK(P3))
        PRED(DEPOT(TRUCK(P3)))=TAIL(DEPOT(TRUCK(P2)))
        SUCC(TAIL(DEPOT(TRUCK(P3))))=DEPOT(TRUCK(P1))
```

```
          PRED(DEPOT(TRUCK(P1)))=TAIL(DEPOT(TRUCK(P3)))
          GOTO 5
        ENDIF
        IF(TRUCK(P1).EQ.TRUCK(P2)) THEN
          SUCC(TAIL(DEPOT(TRUCK(P1))))=DEPOT(TRUCK(P3))
          PRED(DEPOT(TRUCK(P3)))=TAIL(DEPOT(TRUCK(P1)))
          SUCC(TAIL(DEPOT(TRUCK(P3))))=DEPOT(TRUCK(P1))
          PRED(DEPOT(TRUCK(P1)))=TAIL(DEPOT(TRUCK(P3)))
          GOTO 5
        ENDIF
        IF(TRUCK(P2).EQ.TRUCK(P3)) THEN
          SUCC(TAIL(DEPOT(TRUCK(P1))))=DEPOT(TRUCK(P2))
          PRED(DEPOT(TRUCK(P2)))=TAIL(DEPOT(TRUCK(P1)))
          SUCC(TAIL(DEPOT(TRUCK(P2))))=DEPOT(TRUCK(P1))
          PRED(DEPOT(TRUCK(P1)))=TAIL(DEPOT(TRUCK(P2)))
          GOTO 5
        ENDIF
        IF(TRUCK(P1).EQ.TRUCK(P3)) THEN
          SUCC(TAIL(DEPOT(TRUCK(P1))))=DEPOT(TRUCK(P2))
          PRED(DEPOT(TRUCK(P2)))=TAIL(DEPOT(TRUCK(P1)))
          SUCC(TAIL(DEPOT(TRUCK(P2))))=DEPOT(TRUCK(P1))
          PRED(DEPOT(TRUCK(P1)))=TAIL(DEPOT(TRUCK(P2)))
        ENDIF
C
C
      5 IF(NUM.EQ.1) THEN
          SUCC(TALE(1))=HEAD(1)
          PRED(HEAD(1))=TALE(1)
          GAP1=TALE(1)
          GAP2=HEAD(1)
        ENDIF
        IF(NUM.GT.1) THEN
          DO 6 I=1,NUM-1
          SUCC(TALE(I))=HEAD(I+1)
      6   PRED(HEAD(I+1))=TALE(I)
          SUCC(TALE(NUM))=HEAD(1)
          PRED(HEAD(1))=TALE(NUM)
          GAP1=TALE(1)
          GAP2=SUCC(TALE(1))
        ENDIF
C
C
C
C
C       PERFORM THE EXCHANGE
C
        IS1=SUCC(P1)
        IS2=SUCC(P2)
        IS3=SUCC(P3)
        GOTO (10,20,30,40),TYPE
C
C
C
C
C       TYPE 1 EXCHANGE
C
C
     10 LAST=P1
        NEXT=P2
        SUCC(P1)=P2
     11 NODE=NEXT
        IP=PRED(NODE)
        PRED(NODE)=LAST
        IF(NODE.EQ.IS3) GOTO 50
        SUCC(NODE)=IP
        IF(NODE.EQ.IS1) SUCC(NODE)=P3
        IF(NODE.EQ.IS2) SUCC(NODE)=IS3
        LAST=NODE
        NEXT=SUCC(NODE)
        GOTO 11
C
```

```
C
C
C
C     TYPE 2 EXCHANGE
C
C
   20 LAST=P1
      NEXT=IS2
      SUCC(P1)=IS2
      MODE=FWD
   21 NODE=NEXT
      IP=PRED(NODE)
      PRED(NODE)=LAST
      IF(NODE.EQ.IS3) GOTO 50
      IF(MODE.EQ.BACK) SUCC(NODE)=IP
      IF(NODE.EQ.P3) THEN
        SUCC(NODE)=P2
        MODE=BACK
      ENDIF
      IF(NODE.EQ.IS1) SUCC(NODE)=IS3
      LAST=NODE
      NEXT=SUCC(NODE)
      GOTO 21
C
C
C
C
C     TYPE 3 EXCHANGE
C
C
   30 LAST=P1
      NEXT=IS2
      SUCC(P1)=IS2
   31 NODE=NEXT
      PRED(NODE)=LAST
      IF(NODE.EQ.IS3) GOTO 50
      IF(NODE.EQ.P3) SUCC(NODE)=IS1
      IF(NODE.EQ.P2) SUCC(NODE)=IS3
      LAST=NODE
      NEXT=SUCC(NODE)
      GOTO 31
C
C
C
C
C     TYPE 4 EXCHANGE
C
C
   40 LAST=P1
      NEXT=P3
      SUCC(P1)=P3
      MODE=BACK
   41 NODE=NEXT
      IP=PRED(NODE)
      PRED(NODE)=LAST
      IF(NODE.EQ.IS3) GOTO 50
      IF(MODE.EQ.BACK) SUCC(NODE)=IP
      IF(NODE.EQ.IS2) THEN
        SUCC(NODE)=IS1
        MODE=FWD
      ENDIF
      IF(NODE.EQ.P2) SUCC(NODE)=IS3
      LAST=NODE
      NEXT=SUCC(NODE)
      GOTO 41
C
C
C
C
C
C     CALCULATION OF ROUTE LENGTHS AND LOADS.
```

```
C
C
C
   50 CONTINUE
      PRED(IS3)=LAST
      NEXT=FIRST
      INSIDE=0
   51 NODE=NEXT
      IF(NODE.EQ.FIRST.AND.INSIDE.EQ.1) GOTO 60
      INSIDE=1
      IF(NODE.GT.NCITY) THEN
        ITRK=TRUCK(NODE)
        LOAD(ITRK)=0
        LENGTH(ITRK)=0
        CUMLD(NODE)=0
        CUMLN(NODE)=0
      ENDIF
      TRUCK(NODE)=ITRK
      LOAD(ITRK)=LOAD(ITRK)+DEMAND(NODE)
      LENGTH(ITRK)=LENGTH(ITRK)+DIST(NODE,SUCC(NODE))
      IF(SUCC(NODE).LE.NCITY) LENGTH(ITRK)=LENGTH(ITRK)+ALLOW
      IF(NODE.LE.NCITY) THEN
        CUMLD(NODE)=CUMLD(PRED(NODE))+DEMAND(NODE)
        CUMLN(NODE)=CUMLN(PRED(NODE))+DIST(NODE,PRED(NODE))+ALLOW
      ENDIF
      NEXT=SUCC(NODE)
      GOTO 51
C
C
C
C
C
C     RECONNECT THE ROUTES INVOLVED IN THE EXCHANGE BACK INTO THE
C     ORIGINAL NETWORK.
C
C
   60 CONTINUE
      IPRED=PRED(FIRST)
      SUCC(GAP1)=FIRST
      PRED(FIRST)=GAP1
      SUCC(IPRED)=GAP2
      PRED(GAP2)=IPRED
  999 FORMAT(1H ,'EXCHANGE MADE',4I5)
      NEXT=NCITY+1
      INSIDE=0
   76 NODE=NEXT
      IF(NODE.EQ.NCITY+1.AND.INSIDE.EQ.1) THEN
   78 FORMAT(1H ,'ROUTE LENGTHS ARE',12I5)
        RETURN
      ENDIF
      INSIDE=1
   77 FORMAT(1H ,8I5)
      NEXT=SUCC(NODE)
      GOTO 76
      RETURN
      END
C
C*********************************************************************
C
      SUBROUTINE FXCH3(P1,P2,P3,PRED,SUCC,TYPE)
C
C     THIS SUBROUTINE MAKES 'POTENTIAL' 3-ARC EXCHANGES TO A ROUTE
C     STRUCTURE.  NO 'ACTUAL' EXCHANGES ARE MADE.
C
C*********************************************************************
      CHARACTER*44 PNAME
      DOUBLE PRECISION DSEED
      INTEGER P1,P2,P3,FIRST,TAIL(120),TALE(20),HEAD(20),GAP1,GAP2,
     *        TYPE,FWD,BACK,START,END,WTLIM,DISTLM,ALLOW,DLIMIT,
     *        D1,D2,D3,D4,D5,D6,DEPOT(20),PRED(120),SUCC(120),
     *        TRUCK(120),DEMAND(0:120),LENGTH(120),LOAD(120),
```

```
     *        CUMLD(120),CUMLN(120),TEMTRK(120),FEASLD(20),FEASLN(20),
     *        PERMPR(120),PERMSU(120),PERMTR(120),ISORT(6000),
     *        JSORT(6000),DIST(0:120,0:120),XCOORD(0:120),
     *        YCOORD(0:120),LOKK(120),PDUM(120),SDUM(120)
      DIMENSION SORT(6000),TRADE(7,500)
      COMMON IROUTE,NCITY,IRUN,START,END,WTLIM,DISTLM,ALLOW,NPERM,
     *ICOUNT,LDDVLM,LNDVLM,DLIMIT,MAXLD,MINLD,MAXLN,MINLN,D1,D2,D3,D4,
     *D5,D6,DEPOT,PDUM,SDUM,TRUCK,DEMAND,LENGTH,LOAD,CUMLD,CUMLN,TEMTRK,
     *FEASLD,FEASLN,PERMPR,PERMSU,PERMTR,ISORT,JSORT,SORT,DIST,XCOORD,
     *YCOORD,LOKK,TRADE,NTRADE,DSEED
C
C
C
C     NULL EXCHANGE:
C
      IF(P1.EQ.PRED(P2).AND.P2.EQ.PRED(P3).AND.TYPE.EQ.1) RETURN
      IF(TYPE.EQ.3.AND.P1.GT.NCITY.AND.P2.GT.NCITY.AND.P3.GT.NCITY)
     *  RETURN
C
C
C
C
C     REMOVE THE ROUTES INVOLVED IN THE EXCHANGE FROM THE NETWORK AND
C     FORM A SEPARATE NETWORK WITHIN WHICH THE 3-OPT EXCHANGE WILL TAKE
C     PLACE.
C
C
      IS1=SUCC(P1)
      IS2=SUCC(P2)
      IS3=SUCC(P3)
      FWD=0
      BACK=1
      DO 1 I=NCITY+1,NCITY+IROUTE
    1 TAIL(DEPOT(TRUCK(PRED(I))))=PRED(I)
      NEXT=NCITY+1
      NUM=0
      INSIDE=0
    3 NODE=NEXT
      IF(NODE.EQ.NCITY+1.AND.INSIDE.EQ.1) GOTO 4
      INSIDE=1
      IF(TRUCK(NODE).EQ.TRUCK(P1).OR.TRUCK(NODE).EQ.TRUCK(P2).OR.
     *   TRUCK(NODE).EQ.TRUCK(P3)) THEN
        NEXT=SUCC(TAIL(NODE))
        GOTO 3
      ENDIF
      NUM=NUM+1
      HEAD(NUM)=NODE
      TALE(NUM)=TAIL(NODE)
      NEXT=SUCC(TAIL(NODE))
      GOTO 3
    4 IF(TRUCK(P1).EQ.TRUCK(P2).AND.TRUCK(P1).EQ.TRUCK(P3)) THEN
        SUCC(TAIL(DEPOT(TRUCK(P1))))=DEPOT(TRUCK(P1))
        PRED(DEPOT(TRUCK(P1)))=TAIL(DEPOT(TRUCK(P1)))
        GOTO 5
      ENDIF
      IF(TRUCK(P1).NE.TRUCK(P2).AND.TRUCK(P2).NE.TRUCK(P3).AND.
     *   TRUCK(P1).NE.TRUCK(P3)) THEN
        SUCC(TAIL(DEPOT(TRUCK(P1))))=DEPOT(TRUCK(P2))
        PRED(DEPOT(TRUCK(P2)))=TAIL(DEPOT(TRUCK(P1)))
        SUCC(TAIL(DEPOT(TRUCK(P2))))=DEPOT(TRUCK(P3))
        PRED(DEPOT(TRUCK(P3)))=TAIL(DEPOT(TRUCK(P2)))
        SUCC(TAIL(DEPOT(TRUCK(P3))))=DEPOT(TRUCK(P1))
        PRED(DEPOT(TRUCK(P1)))=TAIL(DEPOT(TRUCK(P3)))
        GOTO 5
      ENDIF
      IF(TRUCK(P1).EQ.TRUCK(P2)) THEN
        SUCC(TAIL(DEPOT(TRUCK(P1))))=DEPOT(TRUCK(P3))
        PRED(DEPOT(TRUCK(P3)))=TAIL(DEPOT(TRUCK(P1)))
        SUCC(TAIL(DEPOT(TRUCK(P3))))=DEPOT(TRUCK(P1))
        PRED(DEPOT(TRUCK(P1)))=TAIL(DEPOT(TRUCK(P3)))
        GOTO 5
```

```
      ENDIF
      IF(TRUCK(P2).EQ.TRUCK(P3)) THEN
        SUCC(TAIL(DEPOT(TRUCK(P1))))=DEPOT(TRUCK(P2))
        PRED(DEPOT(TRUCK(P2)))=TAIL(DEPOT(TRUCK(P1)))
        SUCC(TAIL(DEPOT(TRUCK(P2))))=DEPOT(TRUCK(P1))
        PRED(DEPOT(TRUCK(P1)))=TAIL(DEPOT(TRUCK(P2)))
        GOTO 5
      ENDIF
      IF(TRUCK(P1).EQ.TRUCK(P3)) THEN
        SUCC(TAIL(DEPOT(TRUCK(P1))))=DEPOT(TRUCK(P2))
        PRED(DEPOT(TRUCK(P2)))=TAIL(DEPOT(TRUCK(P1)))
        SUCC(TAIL(DEPOT(TRUCK(P2))))=DEPOT(TRUCK(P1))
        PRED(DEPOT(TRUCK(P1)))=TAIL(DEPOT(TRUCK(P2)))
      ENDIF
C
C
    5 IF(NUM.EQ.1) THEN
        SUCC(TALE(1))=HEAD(1)
        PRED(HEAD(1))=TALE(1)
        GAP1=TALE(1)
        GAP2=HEAD(1)
      ENDIF
      IF(NUM.GT.1) THEN
        DO 6 I=1,NUM-1
        SUCC(TALE(I))=HEAD(I+1)
    6   PRED(HEAD(I+1))=TALE(I)
        SUCC(TALE(NUM))=HEAD(1)
        PRED(HEAD(1))=TALE(NUM)
        GAP1=TALE(1)
        GAP2=SUCC(TALE(1))
      ENDIF
C
C
C
C
C     PERFORM THE EXCHANGE
C
      IS1=SUCC(P1)
      IS2=SUCC(P2)
      IS3=SUCC(P3)
      GOTO (10,20,30,40),TYPE
C
C
C
C
C     TYPE 1 EXCHANGE
C
C
   10 LAST=P1
      NEXT=P2
      SUCC(P1)=P2
   11 NODE=NEXT
      IP=PRED(NODE)
      PRED(NODE)=LAST
      IF(NODE.EQ.IS3) GOTO 50
      SUCC(NODE)=IP
      IF(NODE.EQ.IS1) SUCC(NODE)=P3
      IF(NODE.EQ.IS2) SUCC(NODE)=IS3
      LAST=NODE
      NEXT=SUCC(NODE)
      GOTO 11
C
C
C
C
C     TYPE 2 EXCHANGE
C
C
   20 LAST=P1
      NEXT=IS2
      SUCC(P1)=IS2
```

```
      MODE=FWD
   21 NODE=NEXT
      IP=PRED(NODE)
      PRED(NODE)=LAST
      IF(NODE.EQ.IS3) GOTO 50
      IF(MODE.EQ.BACK) SUCC(NODE)=IP
      IF(NODE.EQ.P3) THEN
        SUCC(NODE)=P2
        MODE=BACK
      ENDIF
      IF(NODE.EQ.IS1) SUCC(NODE)=IS3
      LAST=NODE
      NEXT=SUCC(NODE)
      GOTO 21
C
C
C
C
C     TYPE 3 EXCHANGE
C
C
   30 LAST=P1
      NEXT=IS2
      SUCC(P1)=IS2
   31 NODE=NEXT
      PRED(NODE)=LAST
      IF(NODE.EQ.IS3) GOTO 50
      IF(NODE.EQ.P3) SUCC(NODE)=IS1
      IF(NODE.EQ.P2) SUCC(NODE)=IS3
      LAST=NODE
      NEXT=SUCC(NODE)
      GOTO 31
C
C
C
C
C     TYPE 4 EXCHANGE
C
C
   40 LAST=P1
      NEXT=P3
      SUCC(P1)=P3
      MODE=BACK
   41 NODE=NEXT
      IP=PRED(NODE)
      PRED(NODE)=LAST
      IF(NODE.EQ.IS3) GOTO 50
      IF(MODE.EQ.BACK) SUCC(NODE)=IP
      IF(NODE.EQ.IS2) THEN
        SUCC(NODE)=IS1
        MODE=FWD
      ENDIF
      IF(NODE.EQ.P2) SUCC(NODE)=IS3
      LAST=NODE
      NEXT=SUCC(NODE)
      GOTO 41
C
C
C
C
C
C
C
C
   50 CONTINUE
      PRED(IS3)=LAST
C
C
C
C
C
```

```
C     RECONNECT THE ROUTES INVOLVED IN THE EXCHANGE BACK INTO THE
C     ORIGINAL NETWORK.
C
C
      IPRED=PRED(DEPOT(TRUCK(P1)))
      SUCC(GAP1)=DEPOT(TRUCK(P1))
      PRED(DEPOT(TRUCK(P1)))=GAP1
      SUCC(IPRED)=GAP2
      PRED(GAP2)=IPRED
      RETURN
      END
C
C
C*********************************************************************
C
      SUBROUTINE SAVNGS
C
C
C     THIS SUBROUTINE IS USED TO IMPLEMENT THE CONCURRENT VERSION
C     OF CLARKE AND WRIGHT'S SAVINGS ALGORITHM FOR SOLVING THE
C     VEHICLE ROUTING PROBLEM.
C
C*********************************************************************
      CHARACTER*44 PNAME,IPLACE
      INTEGER EUCLID,CITY,XCOORD(0:120),YCOORD(0:120),DEMAND(0:120)
      INTEGER HEAD(120),TAIL(120),PRED(120),SUCC(120),ROUTES,TWGT,WTLIM
      INTEGER DIST,ALLOW,TDIST,DISTLM,TRUCK,IR(100),IFLAG(40),
     * PERMI(40),PERMJ(40)
      DOUBLE PRECISION DSEED
      INTEGER START,END,POINT1,POINT2,D,FEASLD(20),FEASLN(20)
      INTEGER FTRUCK,FWD,BACK,TEMTRK(120),CUMLD(120),CUMLN(120)
      INTEGER FOUND,TRCNT,TRK,TAG,D1,D2,D3,D4,D5,D6,D7,D8
      INTEGER FSTART,FEND,FPRED(120),FSUCC(120)
      INTEGER PERMPR(120),PERMSU(120),PERMTR(120)
      INTEGER DLIMIT,DEPOT(20),LOKK(120)
      DIMENSION DIST(0:120,0:120),SAVING(3,6000),SORT(6000),PERMSV(40)
      DIMENSION ISORT(6000),JSORT(6000),LOAD(120),TRUCK(120)
      DIMENSION LENGTH(120),WORK(6),TRADE(7,500)
      COMMON IROUTE,NCITY,IRUN,START,END,WTLIM,DISTLM,ALLOW,NPERM,
     *ICOUNT,LDDVLM,LNDVLM,DLIMIT,MAXLD,MINLD,MAXLN,MINLN,D1,D2,D3,D4,
     *D5,D6,DEPOT,PRED,SUCC,TRUCK,DEMAND,LENGTH,LOAD,CUMLD,CUMLN,TEMTRK,
     *FEASLD,FEASLN,PERMPR,PERMSU,PERMTR,ISORT,JSORT,SORT,DIST,XCOORD,
     *YCOORD,LOKK,TRADE,NTRADE,DSEED
C
C
C         SET UP HEADERS TO EACH OF NCITY ROUTES.
C         EACH ROUTE IS INITIALLY ROOTED AT DEPOT.
C
C
      DO 10 I=1,NCITY
        HEAD(I)=I
        TAIL(I)=I
        PRED(I)=0
        SUCC(I)=0
        LOAD(I)=DEMAND(I)
        LENGTH(I)=DIST(0,I) + DIST(I,0) + ALLOW
   10   TRUCK(I)=I
      ROUTES=NCITY
C
C
C         START AT TOP OF SAVINGS FILE AND FIND A VALID SAVINGS
C         WHICH CAN BE APPLIED TO A PAIR OF UNLINKED CITIES.
C
C
      NSAV=0
   11 NSAV=NSAV+1
      IF(NSAV.GT.ICOUNT) GOTO 15
C
C         ROUTE LENGTH AND WEIGHT LIMITS CANNOT BE EXCEEDED.
C
      TDIST=FLOAT(LENGTH(TRUCK(ISORT(NSAV)))  +
```

```
     *      LENGTH(TRUCK(JSORT(NSAV)))) - SORT(NSAV) + 0.5
      TWGT=LOAD(TRUCK(ISORT(NSAV))) + LOAD(TRUCK(JSORT(NSAV)))
      IF(TWGT.GT.WTLIM) GOTO 11
      IF(TDIST.GT.DISTLM) GOTO 11
C
C        BOTH CITIES TO BE LINKED MUST BE DIRECTLY CONNECTED TO DEPOT.
C
      IF(PRED(ISORT(NSAV)).NE.O.AND.SUCC(ISORT(NSAV)).NE.O) GOTO 11
      IF(PRED(JSORT(NSAV)).NE.O.AND.SUCC(JSORT(NSAV)).NE.O) GOTO 11
C
C         BOTH CITIES TO BE LINKED CANNOT BE IN SAME SUBTOUR.
C
      IF(TRUCK(ISORT(NSAV)).EQ.TRUCK(JSORT(NSAV))) GOTO 11
C
C         CASE 1: BOTH CITIES ARE AT BEGINNING OF SUBTOUR.
C
      IF(HEAD(TRUCK(ISORT(NSAV))).EQ.ISORT(NSAV).AND.
     *   HEAD(TRUCK(JSORT(NSAV))).EQ.JSORT(NSAV)) THEN
C
        HEAD(TRUCK(ISORT(NSAV)))=TAIL(TRUCK(JSORT(NSAV)))
        HEAD(TRUCK(JSORT(NSAV)))=O
        TAIL(TRUCK(JSORT(NSAV)))=O
        PRED(ISORT(NSAV))=JSORT(NSAV)
        LAST=ISORT(NSAV)
        NODE=JSORT(NSAV)
   12   TRUCK(NODE)=TRUCK(ISORT(NSAV))
        IF(SUCC(NODE).NE.O) THEN
          NEXT=SUCC(NODE)
          PRED(NODE)=NEXT
          SUCC(NODE)=LAST
          LAST=NODE
          NODE=NEXT
          GOTO 12
        END IF
        PRED(NODE)=O
        SUCC(NODE)=LAST
        LENGTH(TRUCK(ISORT(NSAV)))=TDIST
        LOAD(TRUCK(ISORT(NSAV)))=TWGT
        ROUTES=ROUTES-1
        GOTO 11
      END IF
C
C         CASE 2: BOTH CITIES ARE AT END OF SUBTOUR.
C
      IF(TAIL(TRUCK(ISORT(NSAV))).EQ.ISORT(NSAV).AND.
     *   TAIL(TRUCK(JSORT(NSAV))).EQ.JSORT(NSAV)) THEN
        TAIL(TRUCK(ISORT(NSAV)))=HEAD(TRUCK(JSORT(NSAV)))
        TAIL(TRUCK(JSORT(NSAV)))=O
        HEAD(TRUCK(JSORT(NSAV)))=O
        SUCC(ISORT(NSAV))=JSORT(NSAV)
        LAST=ISORT(NSAV)
        NODE=JSORT(NSAV)
   13   TRUCK(NODE)=TRUCK(ISORT(NSAV))
        IF(PRED(NODE).NE.O) THEN
          NEXT=PRED(NODE)
          SUCC(NODE)=NEXT
          PRED(NODE)=LAST
          LAST=NODE
          NODE=NEXT
          GOTO 13
        END IF
        SUCC(NODE)=O
        PRED(NODE)=LAST
        LENGTH(TRUCK(ISORT(NSAV)))=TDIST
        LOAD(TRUCK(ISORT(NSAV)))=TWGT
        ROUTES=ROUTES + 1
        GOTO 11
      END IF
C
C         CASE 3: ONE CITY IS AT BEGINNING OF A SUBTOUR,
C                 OTHER IS AT END OF SUBTOUR.
```

```
C
      I=ISORT(NSAV)
      J=JSORT(NSAV)
      IF(HEAD(TRUCK(I)).NE.I) THEN
        I=JSORT(NSAV)
        J=ISORT(NSAV)
      END IF
      PRED(I)=J
      SUCC(J)=I
      HEAD(TRUCK(I))=HEAD(TRUCK(J))
      TAIL(TRUCK(J))=0
      HEAD(TRUCK(J))=0
      NODE=J
   14 NEXT=PRED(NODE)
      IF(NEXT.NE.0) THEN
        TRUCK(NODE)=TRUCK(I)
        NODE=NEXT
        GOTO 14
      END IF
      TRUCK(NODE)=TRUCK(I)
      LENGTH(TRUCK(I))=TDIST
      LOAD(TRUCK(I))=TWGT
      ROUTES=ROUTES-1
      GOTO 11
C
C
C
   15 IROUTE=0
      DO 17 I=1,NCITY
        IF(HEAD(I).NE.0) THEN
          IROUTE=IROUTE+1
        END IF
   17 CONTINUE
C
C
C         RENUMBER THE ROUTES, THEN REFORM THE ROUTES INTO SINGLE
C         ROUTE WITH ARTIFICIAL DEPOTS INSERTED BETWEEN ADJACENT
C         ORIGINAL ROUTES.
C
C
      DO 18 I=NCITY+1,NCITY+IROUTE
        DEMAND(I)=0
        XCOORD(I)=XCOORD(0)
   18   YCOORD(I)=YCOORD(0)
      DO 19 I=NCITY+1,NCITY+IROUTE
      DO 19 J=I+1,NCITY+IROUTE
        DIST(I,J)=0
   19   DIST(J,I)=0
C
      DO 20 I=1,NCITY
      DO 20 J=NCITY+1,NCITY+IROUTE+1
        DIST(I,J)=DIST(I,0)
   20   DIST(J,I)=DIST(I,J)
C
      I=NCITY
      TRCNT=0
      DO 21 J=1,NCITY
        IF(HEAD(J).NE.0) THEN
          FOUND=J
          TRCNT=TRCNT+1
          I=I+1
          PRED(HEAD(J))=I
          SUCC(I)=HEAD(J)
          SUCC(TAIL(J))=I+1
          PRED(I+1)=TAIL(J)
          TRUCK(I)=TRCNT
          DEPOT(TRCNT)=I
        END IF
   21 CONTINUE
      SUCC(TAIL(FOUND))=NCITY+1
      PRED(NCITY+1)=TAIL(FOUND)
```

```
C
      DO 22 I=1,IROUTE
        LENGTH(I)=0
   22   LOAD(I)=0
      POINT1=NCITY+1
      NODE=POINT1
   23 IF(SUCC(NODE).EQ.NCITY+1) GOTO 24
        IF(DEMAND(NODE).EQ.0.) TRK=TRUCK(NODE)
        TRUCK(NODE)=TRK
        LENGTH(TRUCK(NODE))=LENGTH(TRUCK(NODE))+DIST(NODE,SUCC(NODE))
        IF(SUCC(NODE).LE.NCITY) LENGTH(TRUCK(NODE))=LENGTH(TRUCK(NODE))
     *                                                     + ALLOW
        LOAD(TRUCK(NODE))=LOAD(TRUCK(NODE))+DEMAND(NODE)
      NODE=SUCC(NODE)
      GOTO 23
   24 CONTINUE
      TRUCK(NODE)=TRK
      LENGTH(TRUCK(NODE))=LENGTH(TRUCK(NODE))+DIST(NODE,SUCC(NODE))
      LOAD(TRUCK(NODE))=LOAD(TRUCK(NODE))+DEMAND(NODE)
      RETURN
  202 FORMAT(1H ,T4,I3)
      END
C
C
C*********************************************************************
C
      SUBROUTINE TSP(START,END,PRED,SUCC,LENGTH,DIST,ALLOW)
C
C
C         THIS SUBROUTINE IS USED TO SOLVE THE TRAVELING SALESMAN
C         PROBLEM USING THE 3-OPT EXCHANGE PROCEDURE.
C
C*********************************************************************
C
      CHARACTER*44 IPLACE
      INTEGER START,END,PRED(120),SUCC(120),POINT1,POINT2,POINT3
      INTEGER DIST(0:120,0:120),D1,D2,D3,D4,D5,D6,FWD,BACK
      INTEGER ALLOW, STACK(9),BESTLN,BSTART,BIEND,BESTP(120),BESTS(120)
C
      BESTLN=999999
      IF(END.EQ.START) THEN
        LENGTH=0
        RETURN
      ENDIF
      IF(END.EQ.SUCC(START)) THEN
        LENGTH=DIST(START,END)+DIST(END,SUCC(END))+ALLOW
        RETURN
      END IF
      IF(END.EQ.SUCC(SUCC(START))) THEN
        L1=DIST(START,SUCC(START))+DIST(SUCC(START),END)
     *      +DIST(END,SUCC(END))+2*ALLOW
        L2=DIST(START,END)+DIST(END,SUCC(START))+DIST(SUCC(START),START)
     *      +2*ALLOW
        IF(L1.LE.L2) THEN
          LENGTH=L1
          RETURN
        ENDIF
C       ELSE
          LENGTH=L2
          IP1=SUCC(START)
          ISUCC=SUCC(END)
          SUCC(START)=END
          PRED(END)=START
          SUCC(END)=IP1
          PRED(IP1)=END
          SUCC(IP1)=ISUCC
          PRED(ISUCC)=IP1
          RETURN
      ENDIF
C
      IS=START
```

```
      IE=END
      IPRED=PRED(START)
      ISUCC=SUCC(END)
      FWD=1
      BACK=2
      PRED(START)=END
      SUCC(END)=START
C
C
C         BEGIN ITERATIONS
C
C
      POINT1=END
    5 POINT1=SUCC(POINT1)
      IF(POINT1.EQ.PRED(PRED(END))) GOTO 100
      IS1=SUCC(POINT1)
      POINT2=POINT1
    6 POINT2=SUCC(POINT2)
      IF(POINT2.EQ.PRED(END)) GOTO 5
      IS2=SUCC(POINT2)
      POINT3=POINT2
    7 POINT3=SUCC(POINT3)
      IF(POINT3.EQ.END) GOTO 6
      IS3=SUCC(POINT3)
C
C
C
      D1=DIST(POINT1,IS1)
      D2=DIST(POINT2,IS2)
      D3=DIST(POINT3,IS3)
C
C
C
C         3-OPT TYPE I EXCHANGE
C
C
C
      NTYPE=1
      D4=DIST(POINT1,POINT2)
      D5=DIST(SUCC(POINT1),POINT3)
      D6=DIST(SUCC(POINT2),SUCC(POINT3))
      IF(D1+D2+D3.LE.D4+D5+D6) GOTO 15
C
C
C     ELSE
      LAST=POINT1
      NEXT=POINT2
      SUCC(POINT1)=POINT2
   10 NODE=NEXT
      IP=PRED(NODE)
      PRED(NODE)=LAST
      IF(NODE.EQ.IS3) GOTO 82
      SUCC(NODE)=IP
      IF(NODE.EQ.IS1) SUCC(NODE)=POINT3
      IF(NODE.EQ.IS2) SUCC(NODE)=IS3
      LAST=NODE
      NEXT=SUCC(NODE)
      GOTO 10
C
C
C
C
C
C
C        3-OPT TYPE II EXCHANGE
C
C
C
   15 NTYPE=2
      D4=DIST(POINT1,IS2)
      D5=DIST(POINT3,POINT2)
```

```
      D6=DIST(IS1,IS3)
      IF(D1+D2+D3.LE.D4+D5+D6) GOTO 25
C     ELSE
      LAST=POINT1
      NEXT=IS2
      SUCC(POINT1)=IS2
      MODE=FWD
   20 NODE=NEXT
      IP=PRED(NODE)
      PRED(NODE)=LAST
      IF(NODE.EQ.IS3) GOTO 82
      IF(MODE.EQ.BACK) SUCC(NODE)=IP
      IF(NODE.EQ.POINT3) THEN
        SUCC(NODE)=POINT2
        MODE=BACK
      END IF
      IF(NODE.EQ.IS1) SUCC(NODE)=IS3
      LAST=NODE
      NEXT=SUCC(NODE)
      GOTO 20
C
C
C
C       3-OPT TYPE III EXCHANGE
C
C
C
   25 NTYPE=3
      D4=DIST(POINT1,IS2)
      D5=DIST(POINT3,IS1)
      D6=DIST(POINT2,IS3)
      IF(D1+D2+D3.LE.D4+D5+D6) GOTO 35
C     ELSE
      LAST=POINT1
      NEXT=IS2
      SUCC(POINT1)=IS2
   30 NODE=NEXT
      PRED(NODE)=LAST
      IF(NODE.EQ.IS3) GOTO 82
      IF(NODE.EQ.POINT3) SUCC(NODE)=IS1
      IF(NODE.EQ.POINT2) SUCC(NODE)=IS3
      LAST=NODE
      NEXT=SUCC(NODE)
      GOTO 30
C
C
C
C        3-OPT TYPE IV EXCHANGE
C
C
C
   35 NTYPE=4
      D4=DIST(POINT1,POINT3)
      D5=DIST(IS2,IS1)
      D6=DIST(POINT2,IS3)
      IF(D1+D2+D3.LE.D4+D5+D6) GOTO 45
C     ELSE
      LAST=POINT1
      NEXT=POINT3
      SUCC(POINT1)=POINT3
      MODE=BACK
   40 NODE=NEXT
      IP=PRED(NODE)
      PRED(NODE)=LAST
      IF(NODE.EQ.IS3) GOTO 82
      IF(MODE.EQ.BACK) SUCC(NODE)=IP
      IF(NODE.EQ.IS2) THEN
        SUCC(NODE)=IS1
        MODE=FWD
      END IF
      IF(NODE.EQ.POINT2) SUCC(NODE)=IS3
```

```
      LAST=NODE
      NEXT=SUCC(NODE)
      GOTO 40
C
C
C
C        3-OPT TYPE V EXCHANGE
C
C
C
   45 NTYPE=5
      D4=DIST(POINT1,POINT2)
      D5=DIST(IS1,IS2)
      IF(D1+D2.LE.D4+D5) GOTO 48
C     ELSE
      LAST=POINT1
      NEXT=POINT2
      SUCC(POINT1)=POINT2
   46 NODE=NEXT
      IP=PRED(NODE)
      PRED(NODE)=LAST
      IF(NODE.EQ.IS2) GOTO 82
      SUCC(NODE)=IP
      IF(NODE.EQ.IS1) SUCC(NODE)=IS2
      LAST=NODE
      NEXT=SUCC(NODE)
      GOTO 46
C
C
C
C       3-OPT TYPE VI EXCHANGE
C
C
C
   48 NTYPE=6
      D4=DIST(POINT2,POINT3)
      D5=DIST(IS2,IS3)
      IF(D2+D3.LE.D4+D5) GOTO 54
      LAST=POINT2
      NEXT=POINT3
      SUCC(POINT2)=POINT3
   50 NODE=NEXT
      IP=PRED(NODE)
      PRED(NODE)=LAST
      IF(NODE.EQ.IS3) GOTO 82
      SUCC(NODE)=IP
      IF(NODE.EQ.IS2) SUCC(NODE)=IS3
      LAST=NODE
      NEXT=SUCC(NODE)
      GOTO 50
C
C
C
C      3-OPT TYPE VII EXCHANGE
C
C
C
   54 NTYPE=7
      D4=DIST(POINT1,POINT3)
      D5=DIST(IS1,IS3)
      IF(D1+D3.LE.D4+D5) GOTO 7
      LAST=POINT1
      NEXT=POINT3
      SUCC(POINT1)=POINT3
   55 NODE=NEXT
      IP=PRED(NODE)
      PRED(NODE)=LAST
      IF(NODE.EQ.IS3) GOTO 82
      SUCC(NODE)=IP
      IF(NODE.EQ.IS1) SUCC(NODE)=IS3
      LAST=NODE
```

```
      NEXT=SUCC(NODE)
      GOTO 55
C
C
C         ROTATE
C
C
C
   82 FSTART=END
      END=PRED(END)
      START=FSTART
      POINT1=END
      GOTO 5
C
C
C
C       CALCULATE LENGTH OF ROUTE
C
C
C
  100 START=IS
      IEND=PRED(START)
      END=IEND
      LN=DIST(IEND,ISUCC)
      NEXT=SUCC(START)
  101 NODE=NEXT
      LN=LN+DIST(NODE,PRED(NODE))+ALLOW
      IF(NODE.EQ.IEND) GOTO 102
      NEXT=SUCC(NODE)
      GOTO 101
C
C
C
C     STORE THE BEST TSP SOLUTION
C
C
  102 IF(LN.GE.BESTLN) GOTO 104
C     ELSE
      BESTLN=LN
      BSTART=START
      BIEND=IEND
      BESTS(START)=SUCC(START)
      BESTP(START)=IEND
      NEXT=SUCC(START)
  103 NODE=NEXT
      BESTP(NODE)=PRED(NODE)
      BESTS(NODE)=SUCC(NODE)
      IF(NODE.EQ.IEND) GOTO 104
      NEXT=SUCC(NODE)
      GOTO 103
C
C
C
C     ALTER ROUTE STRUCTURE FOR NEXT TSP SOLUTION
C
C
  104 NTSP=NTSP+1
      IF(NTSP.EQ.1) LN1=LN
      IF(NTSP.EQ.2) LN2=LN
      IF(NTSP.LT.2) THEN
        I1=SUCC(START)
        I2=PRED(END)
        I3=PRED(I2)
        SUCC(START)=I2
        PRED(I2)=START
        SUCC(I2)=I1
        PRED(I1)=I2
        SUCC(I3)=END
        PRED(END)=I3
        POINT1=END
        GOTO 5
```

```
      ENDIF
C
C
C
C     RETRIEVE BEST TSP SOLUTION
C
C
      LENGTH=BESTLN
      START=BSTART
      SUCC(START)=BESTS(START)
      IEND=BIEND
      PRED(START)=BIEND
      NEXT=BESTS(START)
  105 NODE=NEXT
      PRED(NODE)=BESTP(NODE)
      SUCC(NODE)=BESTS(NODE)
      IF(NODE.EQ.IEND) GOTO 106
      NEXT=BESTS(NODE)
      GOTO 105
C
C
C
C     RE-ESTABLISH ROUTE'S POSITION WITHIN OVERALL ROUTE STRUCTURE
C
C
  106 START=IS
      IEND=PRED(START)
      SUCC(IEND)=ISUCC
      PRED(ISUCC)=IEND
      PRED(START)=IPRED
      SUCC(IPRED)=START
      RETURN
      END
C
C
C***********************************************************************
C
      SUBROUTINE FEAS2(P1,P2,FEAS,NTYPE,DIFF,DSTRLX)
C
C
C     THIS SUBROUTINE DETERMINES THE FEASIBILITY OF A 2-OPT EXCHANGE.
C
C***********************************************************************
C
C
      CHARACTER*44 PNAME,IPLACE
      INTEGER FMAXLD,FMINLD,FMAXLN,FMINLN,FEAS,P1,P2,DIFF
      INTEGER EUCLID,CITY,XCOORD(0:120),YCOORD(0:120),DEMAND(0:120)
      INTEGER HEAD(120),TAIL(120),PRED(120),SUCC(120),ROUTES,TWGT,WTLIM
      INTEGER DIST,ALLOW,TDIST,DISTLM,TRUCK,IR(100),IFLAG(40),
     * PERMI(40),PERMJ(40)
      DOUBLE PRECISION DSEED
      INTEGER START,END,POINT1,POINT2,D,FEASLD(20),FEASLN(20)
      INTEGER FTRUCK,FWD,BACK,TEMTRK(120),CUMLD(120),CUMLN(120)
      INTEGER FOUND,TRCNT,TRK,TAG,D1,D2,D3,D4,D5,D6,D7,D8,DEES(6)
      INTEGER FSTART,FEND,FPRED(120),FSUCC(120)
      INTEGER PERMPR(120),PERMSU(120),PERMTR(120)
      INTEGER DLIMIT,DEPOT(20),DSTRLX
      DIMENSION DIST(0:120,0:120),SAVING(3,6000),SORT(6000),PERMSV(40)
      DIMENSION ISORT(6000),JSORT(6000),LOAD(120),TRUCK(120)
      DIMENSION LENGTH(120),WORK(6),LOKK(120),TRADE(7,500)
      COMMON IROUTE,NCITY,IRUN,START,END,WTLIM,DISTLM,ALLOW,NPERM,
     *ICOUNT,LDDVLM,LNDVLM,DLIMIT,MAXLD,MINLD,MAXLN,MINLN,DEES,
     *DEPOT,PRED,SUCC,TRUCK,DEMAND,LENGTH,LOAD,CUMLD,CUMLN,TEMTRK,
     *FEASLD,FEASLN,PERMPR,PERMSU,PERMTR,ISORT,JSORT,SORT,DIST,XCOORD,
     *YCOORD,LOKK,TRADE,NTRADE,DSEED
C
C
      FEAS=0
      LDDEV=MAXLD-MINLD
      LNDEV=MAXLN-MINLN
```

```
C
C
C
C     IF BOTH POINTS ARE IN SAME ROUTE, THE EXCHANGE IS FEASIBLE.
C
      IF(TRUCK(P1).EQ.TRUCK(P2)) THEN
        DO 1 I=1,IROUTE
    1   FEASLN(I)=LENGTH(I)
        FEAS=1
        D1=DIST(P1,SUCC(P1))
        D2=DIST(P2,SUCC(P2))
        D3=DIST(P1,P2)
        D4=DIST(SUCC(P1),SUCC(P2))
        FEASLN(TRUCK(P1))=LENGTH(TRUCK(P1))+D3+D4-D1-D2
        IF(FEASLN(TRUCK(P1)).LT.MINLN) MINLN=FEASLN(TRUCK(P1))
        FMAXLN=-99
        DO 2 I=1,IROUTE
        IF(FEASLN(I).GT.FMAXLN) FMAXLN=FEASLN(I)
    2   CONTINUE
        MAXLN=FMAXLN
        RETURN
      ENDIF
C
C
C
C
C     IF POINTS P1 AND P2 ARE IN DIFFERENT ROUTES, BOTH ROUTES MIGHT
C     BE AFFECTED BY AN EXCHANGE.
C
      DO 3 I=1,IROUTE
        FEASLD(I)=LOAD(I)
    3   FEASLN(I)=LENGTH(I)
C     1ST ROUTE:
C
      IF(CUMLD(P1)+CUMLD(P2).GT.WTLIM) RETURN
      IF(CUMLN(P1)+CUMLN(P2)+DIST(P1,P2).GT.DISTLM) RETURN
C
C
C     2ND ROUTE:
C
      IS1=SUCC(P1)
      IS2=SUCC(P2)
      IF(LOAD(TRUCK(P1))-CUMLD(P1)+LOAD(TRUCK(P2))-CUMLD(P2).GT.WTLIM)
     *   RETURN
      L1=LENGTH(TRUCK(P1))-CUMLN(IS1)+ALLOW
      IF(IS1.GT.NCITY) L1=0
      L2=LENGTH(TRUCK(P2))-CUMLN(IS2)+ALLOW
      IF(IS2.GT.NCITY) L2=0
      IF(L1+L2+DIST(IS1,IS2).GT.DISTLM) RETURN
C
C
C
C
C     ROUTE LENGTH DEVIATION AND ROUTE LOAD DEVIATION TEST.
C
C
      FEASLD(TRUCK(P1))=CUMLD(P1)+CUMLD(P2)
      FEASLN(TRUCK(P1))=CUMLN(P1)+CUMLN(P2)+DIST(P1,P2)
      FEASLD(TRUCK(P2))=LOAD(TRUCK(P1))-CUMLD(P1)+LOAD(TRUCK(P2))
     *                  -CUMLD(P2)
      FEASLN(TRUCK(P2))=L1+L2+DIST(IS1,IS2)
      FMAXLN=-99
      FMINLN=999999
      DO 4 I=1,IROUTE
        IF(LOKK(DEPOT(I)).NE.0) GOTO 4
        IF(FEASLN(I).GT.FMAXLN) FMAXLN=FEASLN(I)
        IF(FEASLN(I).LT.FMINLN.AND.FEASLN(I).NE.0) FMINLN=FEASLN(I)
    4 CONTINUE
      FMAXLD=-99
      FMINLD=999999
      DO 5 I=1,IROUTE
```

```
        IF(LOKK(DEPOT(I)).NE.0) GOTO 5
        IF(FEASLD(I).GT.FMAXLD) FMAXLD=FEASLD(I)
        IF(FEASLD(I).LT.FMINLD.AND.FEASLD(I).NE.0) FMINLD=FEASLD(I)
    5 CONTINUE
C
C
C
C     TRADEOFF ANALYSIS
C
C
      IF(DSTRLX.GT.0) GOTO 6
      IF(NTYPE.EQ.0) GOTO 6
      IF(FMAXLD-FMINLD.GT.LDDVLM.OR.FMAXLN-FMINLN.GT.LNDVLM) THEN
        NTRADE=NTRADE+1
        IF(NTRADE.GT.500) NTRADE=500
        TRADE(1,NTRADE)=-DIFF
        TRADE(2,NTRADE)=FMAXLD-FMINLD-LDDEV
        TRADE(3,NTRADE)=FMAXLN-FMINLN-LNDEV
        TRADE(4,NTRADE)=P1
        TRADE(5,NTRADE)=P2
        TRADE(6,NTRADE)=0.
        TRADE(7,NTRADE)=0.
      ENDIF
    6 IF(FMAXLN-FMINLN.GT.LNDVLM) RETURN
      IF(FMAXLD-FMINLD.GT.LDDVLM) RETURN
C
C
C
C
C
C
C     ELSE
C     EXCHANGE IS FEASIBLE.
C
      FEAS=1
      MAXLD=FMAXLD
      MINLD=FMINLD
      MAXLN=FMAXLN
      MINLN=FMINLN
      RETURN
      END
C
C
C
C
C***********************************************************************
C
      SUBROUTINE LNDV2(LNRLX)
C
C
C     THIS SUBROUTINE IS USED TO MINIMIZE THE MAXIMUM LENGTH DEVIATION
C     IN ROUTE LENGTHS VIA THE TWO-ARC BRANCH EXCHANGE METHOD.
C
C***********************************************************************
C
C
      CHARACTER*1 MODE
      CHARACTER*44 PNAME
      DOUBLE PRECISION DSEED
      INTEGER START,END,WTLIM,DISTLM,ALLOW,DLIMIT,D1,D2,D3,D4,D5,D6,
     *        DEPOT(20),PRED(120),SUCC(120),FPRED(120),FSUCC(120),
     *        TRUCK(120),DEMAND(0:120),LENGTH(120),LOAD(120),CUMLD(120),
     *        CUMLN(120),TEMTRK(120),FEASLD(20),FEASLN(20),PERMPR(120),
     *        PERMSU(120),PERMTR(120),ISORT(6000),JSORT(6000),
     *        DIST(0:120,0:120),XCOORD(0:120),YCOORD(0:120),TLOAD,TDIST,
     *        POINT1,POINT2,FMAXLD,FMINLD,FMAXLN,FMINLN,DEPOT4,TWTLM
      DIMENSION SORT(6000),LOKK(120),TRADE(7,500)
      COMMON IROUTE,NCITY,IRUN,START,END,WTLIM,DISTLM,ALLOW,NPERM,
     *ICOUNT,LDDVLM,LNDVLM,DLIMIT,MAXLD,MINLD,MAXLN,MINLN,D1,D2,D3,D4,
     *D5,D6,DEPOT,PRED,SUCC,TRUCK,DEMAND,LENGTH,LOAD,CUMLD,CUMLN,TEMTRK,
     *FEASLD,FEASLN,PERMPR,PERMSU,PERMTR,ISORT,JSORT,SORT,DIST,XCOORD,
```

```
     *YCOORD,LOKK,TRADE,NTRADE,DSEED
C
C
C
C
      TLOAD=O
      TDIST=O
      START=GGUBFS(DSEED)*NCITY+1
      END=PRED(START)
      TWTLM=MINO(WTLIM,MAXLD+LDDVLM)
      DO 1 I=1,IROUTE
        FEASLD(I)=LOAD(I)
        FEASLN(I)=LENGTH(I)
        TDIST=TDIST+LENGTH(I)
    1   TLOAD=TLOAD+LOAD(I)
      DO 2 I=1,NCITY+IROUTE
        FPRED(I)=PRED(I)
    2   FSUCC(I)=SUCC(I)
C
C
C
C     DETERMINE LOAD AND LENGTH DEVIATIONS.
C
      LDDEV=MAXLD-MINLD
      LNDEV=MAXLN-MINLN
C
C
C
C     BEGIN ITERATIONS
C
C
      POINT1=PRED(START)
    6 POINT1=SUCC(POINT1)
      IF(POINT1.EQ.PRED(PRED(END))) THEN
        RETURN
      ENDIF
      IF(LOKK(POINT1).EQ.1) GOTO 6
      MODE='F'
      POINT2=SUCC(POINT1)
    7 POINT2=SUCC(POINT2)
      IF(POINT2.EQ.END) GOTO 6
      IF(LOKK(POINT2).EQ.1) GOTO 7
      IF(TRUCK(POINT1).NE.TRUCK(POINT2)) MODE='B'
C
C
C
C     DETERMINE WHETHER BOTH ARCS ARE IN THE SAME ROUTE.  IF SO, IGNORE
C     THIS EXCHANGE.
C
C
      IF(TRUCK(POINT1).EQ.TRUCK(POINT2)) GOTO 7
C
C
C
C     DETERMINE WHETHER AT LEAST ONE OF THE ARCS IS IN THE LONG ROUTE
C     OR IN THE SHORT ROUTE.  IF NOT, IGNORE THE EXCHANGE.
C
C
      IF(LOAD(TRUCK(POINT1)).LE.O.OR.LOAD(TRUCK(POINT2)).LE.O)
     *  GOTO 7
      IF(LENGTH(TRUCK(POINT1)).EQ.MAXLN) GOTO 8
      IF(LENGTH(TRUCK(POINT1)).EQ.MINLN) GOTO 8
      IF(LENGTH(TRUCK(POINT2)).EQ.MAXLN) GOTO 8
      IF(LENGTH(TRUCK(POINT2)).EQ.MINLN) GOTO 8
C     ELSE
      GOTO 7
C
C
C
C     LOAD FEASIBILITY TEST
C
```

```
C
    8 IF(CUMLD(POINT1)+CUMLD(POINT2).GT.TWTLM) GOTO 7
      IF(LOAD(TRUCK(POINT1))-CUMLD(POINT1)+LOAD(TRUCK(POINT2))-
     *   CUMLD(POINT2).GT.TWTLM) GOTO 7
      FEASLD(TRUCK(POINT1))=CUMLD(POINT1)+CUMLD(POINT2)
      FEASLD(TRUCK(POINT2))=LOAD(TRUCK(POINT1))-CUMLD(POINT1)+
     *                      LOAD(TRUCK(POINT2))-CUMLD(POINT2)
C
C
C
C     DETERMINE WHETHER LOAD DEVIATION LIMITS HAVE BEEN MAINTAINED.
C
C
      FMAXLD=-99
      FMINLD=999999
      DO 9 I=1,IROUTE
        IF(LOKK(DEPOT(I)).NE.O) GOTO 9
        IF(FEASLD(I).GT.FMAXLD) FMAXLD=FEASLD(I)
        IF(FEASLD(I).LT.FMINLD.AND.FEASLD(I).NE.O) FMINLD=FEASLD(I)
    9 CONTINUE
      IF(FMAXLD-FMINLD.GT.LDDVLM) THEN
        FEASLD(TRUCK(POINT1))=LOAD(TRUCK(POINT1))
        FEASLD(TRUCK(POINT2))=LOAD(TRUCK(POINT2))
        GOTO 7
      ENDIF
C
C
C
C     ESTABLISH POTENTIAL ROUTE STRUCTURE RESULTING FROM EXCHANGE.
C
C
      CALL FXCH2(POINT1,POINT2,FPRED,FSUCC,NULL)
      IF(NULL.EQ.1) THEN
        FEASLD(TRUCK(POINT1))=LOAD(TRUCK(POINT1))
        FEASLD(TRUCK(POINT2))=LOAD(TRUCK(POINT2))
        GOTO 7
      ENDIF
C
C
C
C     SOLVE TSP FOR EACH ROUTE CHANGED BY THE ARC EXCHANGE.
C
C
      ISTRT=DEPOT(TRUCK(POINT1))
      NEXT=FSUCC(ISTRT)
   10 NODE=NEXT
      IF(NODE.GT.NCITY) THEN
        IEND=FPRED(NODE)
        CALL TSP(ISTRT,IEND,FPRED,FSUCC,LNGTH,DIST,ALLOW)
        NTSP=NTSP+1
        IF(LNGTH.GT.DISTLM) GOTO 15
        FEASLN(TRUCK(POINT1))=LNGTH
        GOTO 11
      ENDIF
      NEXT=FSUCC(NODE)
      GOTO 10
   11 ISTRT=DEPOT(TRUCK(POINT2))
      NEXT=FSUCC(ISTRT)
   12 NODE=NEXT
      IF(NODE.GT.NCITY) THEN
        IEND=FPRED(NODE)
        CALL TSP(ISTRT,IEND,FPRED,FSUCC,LNGTH,DIST,ALLOW)
        NTSP=NTSP+1
        IF(LNGTH.GT.DISTLM) GOTO 15
        FEASLN(TRUCK(POINT2))=LNGTH
        GOTO 13
      ENDIF
      NEXT=FSUCC(NODE)
      GOTO 12
C
C
```

```
C
C     DETERMINE WHETHER DISTANCE INCREASE IS WITHIN ACCEPTABLE LIMITS.
C
C
   13 LTDIST=0
      DO 14 I=1,IROUTE
   14 LTDIST=LTDIST+FEASLN(I)
      IF(LTDIST.LE.DLIMIT) GOTO 17
C     ELSE
   15 DO 16 I=NCITY+1,NCITY+IROUTE
        FSUCC(FPRED(I))=SUCC(FPRED(I))
   16   FPRED(I)=PRED(I)
      DO 160 I=1,2
        IF(I.EQ.1) NODE=DEPOT(TRUCK(POINT1))
        IF(I.EQ.2) NODE=DEPOT(TRUCK(POINT2))
        FEASLD(TRUCK(NODE))=LOAD(TRUCK(NODE))
        FEASLN(TRUCK(NODE))=LENGTH(TRUCK(NODE))
        FSUCC(NODE)=SUCC(NODE)
        NEXT=SUCC(NODE)
  116   NODE=NEXT
        IF(NODE.GT.NCITY) GOTO 160
        FPRED(NODE)=PRED(NODE)
        FSUCC(NODE)=SUCC(NODE)
        NEXT=SUCC(NODE)
        GOTO 116
  160   CONTINUE
      GOTO 7
C
C
C
C     DETERMINE WHETHER ROUTE LENGTH DEVIATION IS REDUCED.
C
C
   17 FMAXLN=-99
      FMINLN=999999
      DO 18 I=1,IROUTE
        IF(LOKK(DEPOT(I)).NE.0) GOTO 18
        IF(FEASLN(I).GT.FMAXLN) FMAXLN=FEASLN(I)
        IF(FEASLN(I).LT.FMINLN.AND.FEASLN(I).NE.0) FMINLN=FEASLN(I)
   18 CONTINUE
      IF(FMAXLN-FMINLN.GE.LNDEV+LNRLX) GOTO 15
C     ELSE
C
C
C
C     CHANGE ROUTE STRUCTURE.
C
C
      MAXLN=FMAXLN
      MINLN=FMINLN
      MAXLD=FMAXLD
      MINLD=FMINLD
      LNDEV=MAXLN-MINLN
      LDDEV=MAXLD-MINLD
      DO 19 I=1,NCITY+IROUTE
        PRED(I)=FPRED(I)
   19   SUCC(I)=FSUCC(I)
      NEXT=NCITY+1
      INSIDE=0
   20 NODE=NEXT
      IF(NODE.EQ.NCITY+1.AND.INSIDE.EQ.1) GOTO 21
      INSIDE=1
      IF(NODE.GT.NCITY) THEN
        ITRK=TRUCK(NODE)
        ILD=0
        ILN=0
      ENDIF
      ILD=ILD+DEMAND(NODE)
      IF(NODE.LE.NCITY) ILN=ILN+DIST(NODE,PRED(NODE))+ALLOW
      CUMLD(NODE)=ILD
      CUMLN(NODE)=ILN
```

```
      TRUCK(NODE)=ITRK
      FEASLD(ITRK)=ILD
      NEXT=SUCC(NODE)
      IF(NEXT.GT.NCITY) FEASLN(ITRK)=ILN+DIST(NODE,NEXT)
      GOTO 20
   21 TDIST=0
      TLOAD=0
      DO 22 I=1,IROUTE
        LOAD(I)=FEASLD(I)
        LENGTH(I)=FEASLN(I)
        TLOAD=TLOAD+LOAD(I)
   22   TDIST=TDIST+LENGTH(I)
      NEXT=NCITY+1
      INSIDE=0
C
C
C
C     ROTATE
C
C
      TWTLM=MIN0(WTLIM,MAXLD+LDDVLM)
      LNRLX=0
      ISTART=END
      END=PRED(END)
      START=ISTART
      POINT1=END
      GOTO 6
      END
C
C
C
C
C
C***********************************************************************
C
      SUBROUTINE LDDV2(LDRLX)
C
C
C     THIS SUBROUTINE IS USED TO MINIMIZE THE MAXIMUM LOAD DEVIATION
C     IN ROUTE LOADS VIA THE TWO-ARC BRANCH EXCHANGE METHOD.
C
C***********************************************************************
C
C
      CHARACTER*1 MODE
      CHARACTER*44 PNAME
      DOUBLE PRECISION DSEED
      INTEGER START,END,WTLIM,DISTLM,ALLOW,DLIMIT,D1,D2,D3,D4,D5,D6,
     *        DEPOT(20),PRED(120),SUCC(120),FPRED(120),FSUCC(120),
     *        TRUCK(120),DEMAND(0:120),LENGTH(120),LOAD(120),CUMLD(120),
     *        CUMLN(120),TEMTRK(120),FEASLD(20),FEASLN(20),PERMPR(120),
     *        PERMSU(120),PERMTR(120),ISORT(6000),JSORT(6000),
     *        DIST(0:120,0:120),XCOORD(0:120),YCOORD(0:120),TLOAD,TDIST,
     *        POINT1,POINT2,FMAXLD,FMINLD,FMAXLN,FMINLN,DEPOT4,TWTLM
      DIMENSION SORT(6000),LOKK(120),TRADE(7,500)
      COMMON IROUTE,NCITY,IRUN,START,END,WTLIM,DISTLM,ALLOW,NPERM,
     *ICOUNT,LDDVLM,LNDVLM,DLIMIT,MAXLD,MINLD,MAXLN,MINLN,D1,D2,D3,D4,
     *D5,D6,DEPOT,PRED,SUCC,TRUCK,DEMAND,LENGTH,LOAD,CUMLD,CUMLN,TEMTRK,
     *FEASLD,FEASLN,PERMPR,PERMSU,PERMTR,ISORT,JSORT,SORT,DIST,XCOORD,
     *YCOORD,LOKK,TRADE,NTRADE,DSEED
C
C
C
C
      TLOAD=0
      TDIST=0
      NTSP=0
      START=GGUBFS(DSEED)*NCITY+1
      END=PRED(START)
      DO 1 I=1,IROUTE
        FEASLD(I)=LOAD(I)
```

```
        FEASLN(I)=LENGTH(I)
        TDIST=TDIST+LENGTH(I)
    1   TLOAD=TLOAD+LOAD(I)
      DO 2 I=1,NCITY+IROUTE
        FPRED(I)=PRED(I)
    2   FSUCC(I)=SUCC(I)
C
C
C
C     DETERMINE LOAD AND LENGTH DEVIATIONS.
C
      LNDEV=MAXLN-MINLN
      LDDEV=MAXLD-MINLD
C
C
C
C     BEGIN ITERATIONS
C
C
      POINT1=PRED(START)
    6 POINT1=SUCC(POINT1)
      IF(POINT1.EQ.PRED(PRED(END))) THEN
        RETURN
      ENDIF
      IF(LOKK(POINT1).EQ.1) GOTO 6
      MODE='F'
      POINT2=SUCC(POINT1)
    7 POINT2=SUCC(POINT2)
      IF(POINT2.EQ.END) GOTO 6
      IF(LOKK(POINT2).EQ.1) GOTO 7
      IF(TRUCK(POINT1).NE.TRUCK(POINT2)) MODE='B'
C
C
C
C     DETERMINE WHETHER BOTH ARCS ARE IN THE SAME ROUTE.  IF SO, IGNORE
C     THIS EXCHANGE.
C
C
      IF(TRUCK(POINT1).EQ.TRUCK(POINT2)) GOTO 7
C
C
C
C     DETERMINE WHETHER AT LEAST ONE OF THE ARCS IS IN THE HEAVY ROUTE
C     OR IN THE LIGHT ROUTE.  IF NOT, IGNORE THE EXCHANGE.
C
C
      IF(LOAD(TRUCK(POINT1)).LE.O.OR.LOAD(TRUCK(POINT2)).LE.O)
     *  GOTO 7
      IF(LOAD(TRUCK(POINT1)).EQ.MAXLD) GOTO 8
      IF(LOAD(TRUCK(POINT1)).EQ.MINLD) GOTO 8
      IF(LOAD(TRUCK(POINT2)).EQ.MAXLD) GOTO 8
      IF(LOAD(TRUCK(POINT2)).EQ.MINLD) GOTO 8
C     ELSE
      GOTO 7
C
C
C
C     LOAD FEASIBILITY TEST
C
C
    8 IF(CUMLD(POINT1)+CUMLD(POINT2).GT.TWTLM) GOTO 7
      IF(LOAD(TRUCK(POINT1))-CUMLD(POINT1)+LOAD(TRUCK(POINT2))-
     *   CUMLD(POINT2).GT.TWTLM) GOTO 7
      FEASLD(TRUCK(POINT1))=CUMLD(POINT1)+CUMLD(POINT2)
      FEASLD(TRUCK(POINT2))=LOAD(TRUCK(POINT1))-CUMLD(POINT1)+
     *                      LOAD(TRUCK(POINT2))-CUMLD(POINT2)
C
C
C
C     DETERMINE WHETHER LOAD DEVIATION IS REDUCED.
C
```

```
C
      FMAXLD=-99
      FMINLD=999999
      DO 9 I=1,IROUTE
        IF(LOKK(DEPOT(I)).NE.O) GOTO 9
        IF(FEASLD(I).GT.FMAXLD) FMAXLD=FEASLD(I)
        IF(FEASLD(I).LT.FMINLD.AND.FEASLD(I).NE.O) FMINLD=FEASLD(I)
    9 CONTINUE
      IF(FMAXLD-FMINLD.GE.LDDEV+LDRLX) THEN
        FEASLD(TRUCK(POINT1))=LOAD(TRUCK(POINT1))
        FEASLD(TRUCK(POINT2))=LOAD(TRUCK(POINT2))
        GOTO 7
      ENDIF
C
C
C
C     ESTABLISH POTENTIAL ROUTE STRUCTURE RESULTING FROM EXCHANGE.
C
C
      CALL FXCH2(POINT1,POINT2,FPRED,FSUCC,NULL)
      IF(NULL.EQ.1) THEN
        FEASLD(TRUCK(POINT1))=LOAD(TRUCK(POINT1))
        FEASLD(TRUCK(POINT2))=LOAD(TRUCK(POINT2))
        GOTO 7
      ENDIF
C
C
C
C     SOLVE TSP FOR EACH ROUTE IN THE ARC EXCHANGE.
C
C
      ISTRT=DEPOT(TRUCK(POINT1))
      NEXT=FSUCC(ISTRT)
   10 NODE=NEXT
      IF(NODE.GT.NCITY) THEN
        IEND=FPRED(NODE)
        CALL TSP(ISTRT,IEND,FPRED,FSUCC,LNGTH,DIST,ALLOW)
        IF(LNGTH.GT.DISTLM) GOTO 15
        FEASLN(TRUCK(POINT1))=LNGTH
        GOTO 11
      ENDIF
      NEXT=FSUCC(NODE)
      GOTO 10
   11 ISTRT=DEPOT(TRUCK(POINT2))
      NEXT=FSUCC(ISTRT)
   12 NODE=NEXT
      IF(NODE.GT.NCITY) THEN
        IEND=FPRED(NODE)
        CALL TSP(ISTRT,IEND,FPRED,FSUCC,LNGTH,DIST,ALLOW)
        IF(LNGTH.GT.DISTLM) GOTO 15
        FEASLN(TRUCK(POINT2))=LNGTH
        GOTO 13
      ENDIF
      NEXT=FSUCC(NODE)
      GOTO 12
C
C
C
C     DETERMINE WHETHER DISTANCE INCREASE IS WITHIN ACCEPTABLE LIMITS.
C
C
   13 LTDIST=O
      DO 14 I=1,IROUTE
   14 LTDIST=LTDIST+FEASLN(I)
      IF(LTDIST.LE.DLIMIT) GOTO 17
C     ELSE
   15 DO 16 I=NCITY+1,NCITY+IROUTE
        FSUCC(FPRED(I))=SUCC(FPRED(I))
   16   FPRED(I)=PRED(I)
      DO 160 I=1,2
        IF(I.EQ.1) NODE=DEPOT(TRUCK(POINT1))
```

```
          IF(I.EQ.2) NODE=DEPOT(TRUCK(POINT2))
          FEASLD(TRUCK(NODE))=LOAD(TRUCK(NODE))
          FEASLN(TRUCK(NODE))=LENGTH(TRUCK(NODE))
          FSUCC(NODE)=SUCC(NODE)
          NEXT=SUCC(NODE)
  116     NODE=NEXT
          IF(NODE.GT.NCITY) GOTO 160
          FPRED(NODE)=PRED(NODE)
          FSUCC(NODE)=SUCC(NODE)
          NEXT=SUCC(NODE)
          GOTO 116
  160     CONTINUE
        GOTO 7
C
C
C
C       DETERMINE WHETHER ROUTE LENGTH DEVIATION LIMITS HAVE BEEN
C       MAINTAINED.
C
C
   17 FMAXLN=-99
      FMINLN=999999
      DO 18 I=1,IROUTE
        IF(LOKK(DEPOT(I)).NE.O) GOTO 18
        IF(FEASLN(I).GT.FMAXLN) FMAXLN=FEASLN(I)
        IF(FEASLN(I).LT.FMINLN.AND.FEASLN(I).NE.O) FMINLN=FEASLN(I)
   18 CONTINUE
      IF(FMAXLN-FMINLN.GT.LNDVLM) GOTO 15
C     ELSE
C
C
C
C     CHANGE ROUTE STRUCTURE.
C
C
      MAXLN=FMAXLN
      MINLN=FMINLN
      MAXLD=FMAXLD
      MINLD=FMINLD
      LNDEV=MAXLN-MINLN
      LDDEV=MAXLD-MINLD
      DO 19 I=1,NCITY+IROUTE
        PRED(I)=FPRED(I)
   19   SUCC(I)=FSUCC(I)
      NEXT=NCITY+1
      INSIDE=O
   20 NODE=NEXT
      IF(NODE.EQ.NCITY+1.AND.INSIDE.EQ.1) GOTO 21
      INSIDE=1
      IF(NODE.GT.NCITY) THEN
        ITRK=TRUCK(NODE)
        ILD=O
        ILN=O
      ENDIF
      ILD=ILD+DEMAND(NODE)
      IF(NODE.LE.NCITY) ILN=ILN+DIST(NODE,PRED(NODE))+ALLOW
      CUMLD(NODE)=ILD
      CUMLN(NODE)=ILN
      TRUCK(NODE)=ITRK
      FEASLD(ITRK)=ILD
      NEXT=SUCC(NODE)
      IF(NEXT.GT.NCITY) FEASLN(ITRK)=ILN+DIST(NODE,NEXT)
      GOTO 20
   21 TDIST=O
      TLOAD=O
      DO 22 I=1,IROUTE
        LOAD(I)=FEASLD(I)
        LENGTH(I)=FEASLN(I)
        TLOAD=TLOAD+LOAD(I)
   22   TDIST=TDIST+LENGTH(I)
      NEXT=NCITY+1
```

```
C
C
C
C     ROTATE
C
C
      TWTLM=MINO(WTLIM,MAXLD+LDDEV)
      LDRLX=O
      ISTART=END
      END=PRED(END)
      START=ISTART
      POINT1=END
      GOTO 6
      END
C
C
C
C
C***********************************************************************
C
      SUBROUTINE FXCH2(P1,P2,PRED,SUCC,NULL)
C
C
C     THIS SUBROUTINE MAKES 'POTENTIAL' TWO-ARC EXCHANGES TO A ROUTE
C     STRUCTURE.  NO 'ACTUAL' EXCHANGES ARE MADE.
C
C***********************************************************************
C
C
      CHARACTER*44 PNAME
      DOUBLE PRECISION DSEED
      INTEGER P1,P2,P3,FIRST,TAIL(120),TALE(20),HEAD(20),GAP1,GAP2,
     *        TYPE,FWD,BACK,START,END,WTLIM,DISTLM,ALLOW,DLIMIT,
     *        D1,D2,D3,D4,D5,D6,DEPOT(20),PRED(120),SUCC(120),
     *        TRUCK(120),DEMAND(0:120),LENGTH(120),LOAD(120),
     *        CUMLD(120),CUMLN(120),TEMTRK(120),FEASLD(20),FEASLN(20),
     *        PERMPR(120),PERMSU(120),PERMTR(120),ISORT(6000),
     *        JSORT(6000),DIST(0:120,0:120),XCOORD(0:120),
     *        YCOORD(0:120),LOKK(120),PDUM(120),SDUM(120)
      DIMENSION SORT(6000),TRADE(7,500)
      COMMON IROUTE,NCITY,IRUN,START,END,WTLIM,DISTLM,ALLOW,NPERM,
     *ICOUNT,LDDVLM,LNDVLM,DLIMIT,MAXLD,MINLD,MAXLN,MINLN,D1,D2,D3,D4,
     *D5,D6,DEPOT,PDUM,SDUM,TRUCK,DEMAND,LENGTH,LOAD,CUMLD,CUMLN,TEMTRK,
     *FEASLD,FEASLN,PERMPR,PERMSU,PERMTR,ISORT,JSORT,SORT,DIST,XCOORD,
     *YCOORD,LOKK,TRADE,NTRADE,DSEED
C
C
C
C     NULL EXCHANGE:
C
      NULL=1
      IF(P1.EQ.PRED(P2).OR.P1.EQ.SUCC(P2)) RETURN
      IF(P1.GT.NCITY.AND.SUCC(P2).GT.NCITY) RETURN
      IF(P2.GT.NCITY.AND.SUCC(P1).GT.NCITY) RETURN
      NULL=0
C
C
C
C     REMOVE THE ROUTES INVOLVED IN THE EXCHANGE FROM THE NETWORK AND
C     FORM A SEPARATE NETWORK WITHIN WHICH THE 2-ARC EXCHANGE WILL TAKE
C     PLACE.
C
C
      IS1=SUCC(P1)
      IS2=SUCC(P2)
      DO 1 I=NCITY+1,NCITY+IROUTE
    1 TAIL(DEPOT(TRUCK(PRED(I))))=PRED(I)
      NEXT=NCITY+1
      NUM=0
      INSIDE=0
    3 NODE=NEXT
```

```
      IF(NODE.EQ.NCITY+1.AND.INSIDE.EQ.1) GOTO 4
      INSIDE=1
      IF(TRUCK(NODE).EQ.TRUCK(P1).OR.TRUCK(NODE).EQ.TRUCK(P2)) THEN
        NEXT=SUCC(TAIL(NODE))
        GOTO 3
      ENDIF
      NUM=NUM+1
      HEAD(NUM)=NODE
      TALE(NUM)=TAIL(NODE)
      NEXT=SUCC(TAIL(NODE))
      GOTO 3
C
    4 IF(TRUCK(P1).EQ.TRUCK(P2)) THEN
        SUCC(TAIL(DEPOT(TRUCK(P1))))=DEPOT(TRUCK(P1))
        PRED(DEPOT(TRUCK(P1)))=TAIL(DEPOT(TRUCK(P1)))
        GOTO 5
      ENDIF
        SUCC(TAIL(DEPOT(TRUCK(P1))))=DEPOT(TRUCK(P2))
        PRED(DEPOT(TRUCK(P2)))=TAIL(DEPOT(TRUCK(P1)))
        SUCC(TAIL(DEPOT(TRUCK(P2))))=DEPOT(TRUCK(P1))
        PRED(DEPOT(TRUCK(P1)))=TAIL(DEPOT(TRUCK(P2)))
C
    5 IF(NUM.EQ.1) THEN
        SUCC(TALE(1))=HEAD(1)
        PRED(HEAD(1))=TALE(1)
        GAP1=TALE(1)
        GAP2=HEAD(1)
      ENDIF
      IF(NUM.GT.1) THEN
        DO 6 I=1,NUM-1
        SUCC(TALE(I))=HEAD(I+1)
    6   PRED(HEAD(I+1))=TALE(I)
        SUCC(TALE(NUM))=HEAD(1)
        PRED(HEAD(1))=TALE(NUM)
        GAP1=TALE(1)
        GAP2=SUCC(TALE(1))
      ENDIF
C
C
C
C     PERFORM THE EXCHANGE.
C
C
      IS1=SUCC(P1)
      IS2=SUCC(P2)
      LAST=P1
      NEXT=P2
      SUCC(P1)=P2
    7 NODE=NEXT
      IP=PRED(NODE)
      PRED(NODE)=LAST
      IF(NODE.EQ.IS2) GOTO 8
      SUCC(NODE)=IP
      IF(NODE.EQ.IS1) SUCC(NODE)=IS2
      LAST=NODE
      NEXT=SUCC(NODE)
      GOTO 7
C
C
C
C     RECONNECT ROUTES INVOLVED IN EXCHANGE BACK INTO ORIGINAL NETWORK.
C
C
    8 PRED(IS2)=LAST
      IPRED=PRED(DEPOT(TRUCK(P1)))
      SUCC(GAP1)=DEPOT(TRUCK(P1))
      PRED(DEPOT(TRUCK(P1)))=GAP1
      SUCC(IPRED)=GAP2
      PRED(GAP2)=IPRED
      RETURN
      END
```

```
C
C
C*********************************************************************
C
      SUBROUTINE LNDV3(LNRLX)
C
C
C     THIS SUBROUTINE IS USED TO MINIMIZE THE MAXIMUM LENGTH DEVIATION
C     IN ROUTE LENGTHS VIA THE THREE-ARC BRANCH EXCHANGE METHOD.
C
C*********************************************************************
C
C
      CHARACTER*1 MODE
      CHARACTER*44 PNAME
      DOUBLE PRECISION DSEED
      INTEGER START,END,WTLIM,DISTLM,ALLOW,DLIMIT,D1,D2,D3,D4,D5,D6,
     *        DEPOT(20),PRED(120),SUCC(120),FPRED(120),FSUCC(120),
     *        TRUCK(120),DEMAND(0:120),LENGTH(120),LOAD(120),CUMLD(120),
     *        CUMLN(120),TEMTRK(120),FEASLD(20),FEASLN(20),PERMPR(120),
     *        PERMSU(120),PERMTR(120),ISORT(6000),JSORT(6000),
     *        DIST(0:120,0:120),XCOORD(0:120),YCOORD(0:120),TLOAD,TDIST,
     *        P1,P2,P3,FMAXLD,FMINLD,FMAXLN,FMINLN,DEPOT4,TR(3),
     *        TRY(120,120),FLDDEV,TWTLM
      DIMENSION SORT(6000),LOKK(120)
      DIMENSION TRADE(7,500)
      COMMON IROUTE,NCITY,IRUN,START,END,WTLIM,DISTLM,ALLOW,NPERM,
     *ICOUNT,LDDVLM,LNDVLM,DLIMIT,MAXLD,MINLD,MAXLN,MINLN,D1,D2,D3,D4,
     *D5,D6,DEPOT,PRED,SUCC,TRUCK,DEMAND,LENGTH,LOAD,CUMLD,CUMLN,TEMTRK,
     *FEASLD,FEASLN,PERMPR,PERMSU,PERMTR,ISORT,JSORT,SORT,DIST,XCOORD,
     *YCOORD,LOKK,TRADE,NTRADE,DSEED
C
C
C
      NTRADE=0
      MODE='F'
      TLOAD=0
      TDIST=0
      START=GGUBFS(DSEED)*NCITY+1
      END=PRED(START)
      TWTLM=MIN0(WTLIM,MAXLD+LDDVLM)
      DO 1 I=1,IROUTE
        FEASLD(I)=LOAD(I)
        FEASLN(I)=LENGTH(I)
        TDIST=TDIST+LENGTH(I)
    1   TLOAD=TLOAD+LOAD(I)
      DO 2 I=1,NCITY+IROUTE
        FPRED(I)=PRED(I)
    2   FSUCC(I)=SUCC(I)
      DO 3 I=1,NCITY+IROUTE
      DO 3 J=1,NCITY+IROUTE
    3   TRY(I,J)=0
C
C
C
C     DETERMINE LOAD AND LENGTH DEVIATIONS
C
C
      LNDEV=MAXLN-MINLN
      LDDEV=MAXLD-MINLD
C
C
C
C     BEGIN ITERATIONS
C
C
      P1=PRED(START)
    6 P1=SUCC(P1)
      IF(P1.EQ.PRED(PRED(END))) THEN
        RETURN
      ENDIF
```

```
      IF(LOKK(P1).EQ.1) GOTO 6
      P2=P1
    7 P2=SUCC(P2)
      IF(P2.EQ.PRED(END)) GOTO 6
      IF(LOKK(P2).EQ.1) GOTO 7
      P3=P2
    8 P3=SUCC(P3)
      IF(P3.EQ.END) GOTO 7
      IF(LOKK(P3).EQ.1) GOTO 8
C
C
C
C     IF ALL ARCS ARE IN THE SAME ROUTE, IGNORE THE EXCHANGE
C
C
      IF(TRUCK(P1).EQ.TRUCK(P2).AND.TRUCK(P1).EQ.TRUCK(P3)) GOTO 8
C
C
C
C     DETERMINE WHETHER AT LEAST ONE OF THE ARCS IS IN THE LONG ROUTE
C     OR IN THE SHORT ROUTE.  IF NOT, IGNORE THE EXCHANGE.
C
C
      IF(LOAD(TRUCK(P1)).LE.O.OR.LOAD(TRUCK(P2)).LE.O.OR.
     *   LOAD(TRUCK(P3)).LE.O) GOTO 8
      IF(LENGTH(TRUCK(P1)).EQ.MAXLN.OR.LENGTH(TRUCK(P1)).EQ.MINLN)
     *   GOTO 99
      IF(LENGTH(TRUCK(P2)).EQ.MAXLN.OR.LENGTH(TRUCK(P2)).EQ.MINLN)
     *   GOTO 99
      IF(LENGTH(TRUCK(P3)).EQ.MAXLN.OR.LENGTH(TRUCK(P3)).EQ.MINLN)
     *   GOTO 99
C     ELSE
      GOTO 8
C
C
   99 CONTINUE
      IF(LNRLX.GT.O) THEN
        IF(P1.GT.NCITY) GOTO 8
        IF(P2.GT.NCITY) GOTO 8
        IF(P3.GT.NCITY) GOTO 8
      ENDIF
  809 FORMAT(1H ,3I7)
      IF(TRUCK(P1).NE.TRUCK(P2).AND.TRUCK(P1).NE.TRUCK(P3).AND.
     *   TRUCK(P3).NE.TRUCK(P2)) THEN
        LONER=O
        GOTO 9
      ENDIF
      IF(TRUCK(P2).EQ.TRUCK(P3)) THEN
        LONER=1
        GOTO 11
      ENDIF
      IF(TRUCK(P1).EQ.TRUCK(P3)) THEN
        LONER=2
        GOTO 11
      ENDIF
      IF(TRUCK(P1).EQ.TRUCK(P2)) THEN
        LONER=3
        GOTO 11
      ENDIF
C
C
C
C     TYPE I EXCHANGE - LOAD FEASIBILITY TEST
C
C
    9 NTYPE=1
        IF(CUMLD(P1)+CUMLD(P2).GT.TWTLM) GOTO 10
        IF(LOAD(TRUCK(P1))-CUMLD(P1)+CUMLD(P3).GT.TWTLM) GOTO 10
        IF(LOAD(TRUCK(P2))-CUMLD(P2)+LOAD(TRUCK(P3))-CUMLD(P3).GT.TWTLM)
     *    GOTO 10
        FEASLD(TRUCK(P1))=CUMLD(P1)+CUMLD(P2)
```

```
      FEASLD(TRUCK(P2))=LOAD(TRUCK(P1))-CUMLD(P1)+CUMLD(P3)
      FEASLD(TRUCK(P3))=LOAD(TRUCK(P2))-CUMLD(P2)+LOAD(TRUCK(P3))
     *                  -CUMLD(P3)
      NUMTRK=3
      TR(1)=P1
      TR(2)=P2
      TR(3)=P3
      GOTO 50
C
C
C
C    TYPE II EXCHANGE - LOAD FEASIBILITY TEST
C
C
   10 NTYPE=2
      IF(CUMLD(P1)+LOAD(TRUCK(P2))-CUMLD(P2).GT.TWTLM) GOTO 11
      IF(CUMLD(P2)+CUMLD(P3).GT.TWTLM) GOTO 11
      IF(LOAD(TRUCK(P1))-CUMLD(P1)+LOAD(TRUCK(P3))-CUMLD(P3).GT.TWTLM)
     *  GOTO 11
      FEASLD(TRUCK(P1))=CUMLD(P1)+LOAD(TRUCK(P2))-CUMLD(P2)
      FEASLD(TRUCK(P2))=CUMLD(P2)+CUMLD(P3)
      FEASLD(TRUCK(P3))=LOAD(TRUCK(P1))-CUMLD(P1)+LOAD(TRUCK(P3))
     *                  -CUMLD(P3)
      NUMTRK=3
      TR(1)=P1
      TR(2)=P2
      TR(3)=P3
      GOTO 50
C
C
C    TYPE III EXCHANGE - LOAD FEASIBILITY TEST
C
C
   11 NTYPE=3
     IF(LONER.EQ.0) THEN
      IF(P1.GT.NCITY.AND.P2.GT.NCITY.AND.P3.GT.NCITY) GOTO 12
      IF(CUMLD(P1)+LOAD(TRUCK(P2))-CUMLD(P2).GT.TWTLM) GOTO 12
      IF(LOAD(TRUCK(P1))-CUMLD(P1)+CUMLD(P3).GT.TWTLM) GOTO 12
      IF(CUMLD(P2)+LOAD(TRUCK(P3))-CUMLD(P3).GT.TWTLM) GOTO 12
      FEASLD(TRUCK(P1))=CUMLD(P1)+LOAD(TRUCK(P2))-CUMLD(P2)
      FEASLD(TRUCK(P2))=LOAD(TRUCK(P1))-CUMLD(P1)+CUMLD(P3)
      FEASLD(TRUCK(P3))=CUMLD(P2)+LOAD(TRUCK(P3))-CUMLD(P3)
      NUMTRK=3
      TR(1)=P1
      TR(2)=P2
      TR(3)=P3
      GOTO 50
     ENDIF
     IF(LONER.EQ.1) THEN
      IF(LOAD(TRUCK(P1))+CUMLD(P3)-CUMLD(P2).GT.TWTLM) GOTO 13
      FEASLD(TRUCK(P1))=LOAD(TRUCK(P1))+CUMLD(P3)-CUMLD(P2)
      FEASLD(TRUCK(P2))=LOAD(TRUCK(P2))-CUMLD(P3)+CUMLD(P2)
      NUMTRK=2
      TR(1)=P1
      TR(2)=P2
      GOTO 50
     ENDIF
     IF(LONER.EQ.3) THEN
      IF(LOAD(TRUCK(P3))+CUMLD(P2)-CUMLD(P1).GT.TWTLM) GOTO 14
      FEASLD(TRUCK(P2))=LOAD(TRUCK(P2))-CUMLD(P2)+CUMLD(P1)
      FEASLD(TRUCK(P3))=LOAD(TRUCK(P3))+CUMLD(P2)-CUMLD(P1)
      NUMTRK=2
      TR(1)=P2
      TR(2)=P3
      GOTO 50
     ENDIF
     IF(LONER.EQ.2) THEN
      IF(LOAD(TRUCK(P2))+CUMLD(P1)-CUMLD(P3).GT.TWTLM) GOTO 13
      FEASLD(TRUCK(P2))=LOAD(TRUCK(P2))+CUMLD(P1)-CUMLD(P3)
      FEASLD(TRUCK(P1))=LOAD(TRUCK(P1))-CUMLD(P1)+CUMLD(P3)
      NUMTRK=2
```

```
        TR(1)=P2
        TR(2)=P1
        GOTO 50
      ENDIF
C
C
C
C     TYPE IV EXCHANGE - LOAD FEASIBILITY TEST
C
C
   12 NTYPE=4
        IF(CUMLD(P1)+CUMLD(P3).GT.TWTLM) GOTO 13
        IF(LOAD(TRUCK(P1))-CUMLD(P1)+LOAD(TRUCK(P2))-CUMLD(P2).GT.TWTLM)
     *    GOTO 13
        IF(CUMLD(P2)+LOAD(TRUCK(P3))-CUMLD(P3).GT.TWTLM) GOTO 13
        FEASLD(TRUCK(P1))=CUMLD(P1)+CUMLD(P3)
        FEASLD(TRUCK(P2))=LOAD(TRUCK(P1))-CUMLD(P1)+LOAD(TRUCK(P2))
     *                    -CUMLD(P2)
        FEASLD(TRUCK(P3))=CUMLD(P2)+LOAD(TRUCK(P3))-CUMLD(P3)
        NUMTRK=3
        TR(1)=P1
        TR(2)=P2
        TR(3)=P3
        GOTO 50
C
C
C
C     TYPE V EXCHANGE - LOAD FEASIBILITY TEST
C
C
   13 NTYPE=5
      IF(TRY(P1,P2).EQ.1) GOTO 14
      TRY(P1,P2)=1
      IF(LONER.EQ.3) GOTO 14
      IF(CUMLD(P1)+CUMLD(P2).GT.TWTLM) GOTO 14
      IF(LOAD(TRUCK(P1))-CUMLD(P1)+LOAD(TRUCK(P2))-CUMLD(P2).GT.TWTLM)
     *  GOTO 14
      FEASLD(TRUCK(P1))=CUMLD(P1)+CUMLD(P2)
      FEASLD(TRUCK(P2))=LOAD(TRUCK(P1))-CUMLD(P1)+LOAD(TRUCK(P2))
     *                  -CUMLD(P2)
      NUMTRK=2
      TR(1)=P1
      TR(2)=P2
      GOTO 50
C
C
C
C     TYPE VI EXCHANGE - LOAD FEASIBILITY TEST
C
C
   14 NTYPE=6
      IF(TRY(P2,P3).EQ.1) GOTO 15
      TRY(P2,P3)=1
      IF(LONER.EQ.1) GOTO 15
      IF(CUMLD(P2)+CUMLD(P3).GT.TWTLM) GOTO 15
      IF(LOAD(TRUCK(P2))-CUMLD(P2)+LOAD(TRUCK(P3))-CUMLD(P3).GT.TWTLM)
     *  GOTO 15
      FEASLD(TRUCK(P2))=CUMLD(P2)+CUMLD(P3)
      FEASLD(TRUCK(P3))=LOAD(TRUCK(P2))-CUMLD(P2)+LOAD(TRUCK(P3))
     *                  -CUMLD(P3)
      NUMTRK=2
      TR(1)=P2
      TR(2)=P3
      GOTO 50
C
C
C
C     TYPE VII EXCHANGE - LOAD FEASIBILITY TEST
C
C
   15 NTYPE=7
```

```
      IF(TRY(P1,P3).EQ.1) GOTO 8
      TRY(P1,P3)=1
      IF(LONER.EQ.2) GOTO 8
      IF(CUMLD(P1)+CUMLD(P3).GT.TWTLM) GOTO 8
      IF(LOAD(TRUCK(P1))-CUMLD(P1)+LOAD(TRUCK(P3))-CUMLD(P3).GT.TWTLM)
     *  GOTO 8
      FEASLD(TRUCK(P1))=CUMLD(P1)+CUMLD(P3)
      FEASLD(TRUCK(P3))=LOAD(TRUCK(P1))-CUMLD(P1)+LOAD(TRUCK(P3))
     *                   -CUMLD(P3)
      NUMTRK=2
      TR(1)=P1
      TR(2)=P3
      GOTO 50
C
C
C
C     DETERMINE WHETHER LOAD DEVIATION LIMITS HAVE BEEN MAINTAINED
C
C
   50 FMAXLD=-99
      FMINLD=999999
      DO 51 I=1,IROUTE
        IF(LOKK(DEPOT(I)).NE.O) GOTO 51
        IF(FEASLD(I).GT.FMAXLD) FMAXLD=FEASLD(I)
        IF(FEASLD(I).LT.FMINLD.AND.FEASLD(I).NE.O) FMINLD=FEASLD(I)
   51 CONTINUE
      IF(LNRLX.GT.O.AND.FMAXLD-FMINLD.GT.LDDVLM) GOTO 1111
C
C
C
C     ESTABLISH POTENTIAL ROUTE STRUCTURE RESULTING FROM EXCHANGE
C
C
      IF(NTYPE.LE.4) CALL FXCH3(P1,P2,P3,FPRED,FSUCC,NTYPE)
      IF(NTYPE.EQ.5) CALL FXCH2(P1,P2,FPRED,FSUCC,NULL)
      IF(NTYPE.EQ.6) CALL FXCH2(P2,P3,FPRED,FSUCC,NULL)
      IF(NTYPE.EQ.7) CALL FXCH2(P1,P3,FPRED,FSUCC,NULL)
      IF(NTYPE.GT.4.AND.NULL.EQ.1) THEN
 1111   FEASLD(TRUCK(P1))=LOAD(TRUCK(P1))
        FEASLD(TRUCK(P2))=LOAD(TRUCK(P2))
        FEASLD(TRUCK(P3))=LOAD(TRUCK(P3))
        IF (LONER.EQ.O) GOTO (10,11,12,13,14,15,8), NTYPE
        IF(LONER.EQ.1) GOTO (11,11,13,13,15,15,8), NTYPE
        IF(LONER.EQ.2) GOTO (11,11,13,13,14,8,8), NTYPE
        IF(LONER.EQ.3) GOTO (11,11,14,14,14,15,8),NTYPE
      ENDIF
C
C
C
C     SOLVE TSP FOR EACH ROUTE AFFECTED BY THE EXCHANGE
C
C
      DO 55 I=1,NUMTRK
        ISTRT=DEPOT(TRUCK(TR(I)))
        NEXT=FSUCC(ISTRT)
   52   NODE=NEXT
        IF(NODE.GT.NCITY) THEN
          IEND=FPRED(NODE)
          CALL TSP(ISTRT,IEND,FPRED,FSUCC,LNGTH,DIST,ALLOW)
          IF(LNGTH.GT.DISTLM) GOTO 58
          FEASLN(TRUCK(TR(I)))=LNGTH
          GOTO 55
        ENDIF
        NEXT=FSUCC(NODE)
        GOTO 52
   55 CONTINUE
C
C
C
C     DETERMINE  DISTANCE INCREASE.
C
```

```
C
      LTDIST=0
      DO 56 I=1,IROUTE
   56 LTDIST=LTDIST+FEASLN(I)
C
C
C
C     DETERMINE WHETHER ROUTE LENGTH DEVIATION IS REDUCED
C
C
      FMAXLN=-99
      FMINLN=999999
      DO 57 I=1,IROUTE
        IF(LOKK(DEPOT(I)).NE.0) GOTO 57
        IF(FEASLN(I).GT.FMAXLN) FMAXLN=FEASLN(I)
        IF(FEASLN(I).LT.FMINLN.AND.FEASLN(I).NE.0) FMINLN=FEASLN(I)
   57 CONTINUE
      IF(FMAXLN-FMINLN.GE.LNDEV+LNRLX) THEN
   58   DO 59 I=NCITY+1,NCITY+IROUTE
          FSUCC(FPRED(I))=SUCC(FPRED(I))
   59     FPRED(I)=PRED(I)
        DO 61 I=1,NUMTRK
          FEASLD(TRUCK(TR(I)))=LOAD(TRUCK(TR(I)))
          FEASLN(TRUCK(TR(I)))=LENGTH(TRUCK(TR(I)))
          NODE=DEPOT(TRUCK(TR(I)))
          FPRED(NODE)=PRED(NODE)
          FSUCC(NODE)=SUCC(NODE)
          NEXT=SUCC(NODE)
   60     NODE=NEXT
          IF(NODE.GT.NCITY) GOTO 61
          FPRED(NODE)=PRED(NODE)
          FSUCC(NODE)=SUCC(NODE)
          NEXT=SUCC(NODE)
          GOTO 60
   61   CONTINUE
        IF(LONER.EQ.0) GOTO (10,11,12,13,14,15,8), NTYPE
        IF(LONER.EQ.1) GOTO (11,11,13,13,15,15,8), NTYPE
        IF(LONER.EQ.2) GOTO (11,11,13,13,14,8,8),  NTYPE
        IF(LONER.EQ.3) GOTO (11,11,14,14,14,15,8), NTYPE
      ENDIF
C
C
C
C     TRADEOFF ANALYSIS
C
C
      IF(LNRLX.GT.0) GOTO 62
      IF(LTDIST.GT.DLIMIT.OR.FMAXLD-FMINLD.GT.LDDVLM) THEN
        NTRADE=NTRADE+1
        IF(NTRADE.GT.500) NTRADE=500
        TRADE(1,NTRADE)=LNDEV-(FMAXLN-FMINLN)
        TRADE(1,NTRADE)=-TRADE(1,NTRADE)
        TRADE(2,NTRADE)=LTDIST-TDIST
        TRADE(3,NTRADE)=FMAXLD-FMINLD-LDDEV
        TRADE(4,NTRADE)=P1
        TRADE(5,NTRADE)=P2
        TRADE(6,NTRADE)=P3
        TRADE(7,NTRADE)=NTYPE
      ENDIF
   62 IF(LTDIST.GT.DLIMIT) GOTO 58
      IF(FMAXLD-FMINLD.GT.LDDVLM) GOTO 58
C     ELSE
C
C
C
C     CHANGE ROUTE STRUCTURE
C
C
      MAXLN=FMAXLN
      MINLN=FMINLN
      MAXLD=FMAXLD
```

```
      MINLD=FMINLD
      LNDEV=MAXLN-MINLN
      LDDEV=MAXLD-MINLD
      DO 66 I=1,NCITY+IROUTE
        PRED(I)=FPRED(I)
   66   SUCC(I)=FSUCC(I)
      NEXT=NCITY+1
      INSIDE=0
   67 NODE=NEXT
      IF(NODE.EQ.NCITY+1.AND.INSIDE.EQ.1) GOTO 68
      INSIDE=1
      IF(NODE.GT.NCITY) THEN
        ITRK=TRUCK(NODE)
        ILD=0
        ILN=0
      ENDIF
      ILD=ILD+DEMAND(NODE)
      IF(NODE.LE.NCITY) ILN=ILN+DIST(NODE,PRED(NODE))+ALLOW
      CUMLD(NODE)=ILD
      CUMLN(NODE)=ILN
      TRUCK(NODE)=ITRK
      FEASLD(ITRK)=ILD
      NEXT=SUCC(NODE)
      IF(NEXT.GT.NCITY) FEASLN(ITRK)=ILN+DIST(NODE,NEXT)
      GOTO 67
   68 TDIST=0
      TLOAD=0
      DO 69 I=1,IROUTE
        LOAD(I)=FEASLD(I)
        LENGTH(I)=FEASLN(I)
        TLOAD=TLOAD+LOAD(I)
   69   TDIST=TDIST+LENGTH(I)
C
C
C
C     ROTATE
C
C
      TWTLM=MIN0(WTLIM,MAXLD+LDDVLM)
      IRLX=LNRLX
      LNRLX=0
      NTRADE=0
      ISTART=END
      IF(IRLX.GT.0) ISTART=DEPOT(TRUCK(PRED(ISTART)))
      END=PRED(ISTART)
      START=ISTART
      P1=END
      DO 70 I=1,NCITY+IROUTE
      DO 70 J=1,NCITY+IROUTE
   70 TRY(I,J)=0
      GOTO 6
      END
C
C
C
C**********************************************************************
C
      SUBROUTINE LDDV3(LDRLX)
C
C
C     THIS SUBROUTINE IS USED TO MINIMIZE THE MAXIMUM LOAD DEVIATION
C     IN ROUTE LOADS VIA THE THREE-ARC BRANCH EXCHANGE METHOD.
C
C**********************************************************************
C
C
      CHARACTER*1 MODE
      CHARACTER*44 PNAME
      DOUBLE PRECISION DSEED
      INTEGER START,END,WTLIM,DISTLM,ALLOW,DLIMIT,D1,D2,D3,D4,D5,D6,
     *        DEPOT(20),PRED(120),SUCC(120),FPRED(120),FSUCC(120),
```

```
     *         TRUCK(120),DEMAND(0:120),LENGTH(120),LOAD(120),CUMLD(120),
     *         CUMLN(120),TEMTRK(120),FEASLD(20),FEASLN(20),PERMPR(120),
     *         PERMSU(120),PERMTR(120),ISORT(6000),JSORT(6000),
     *         DIST(0:120,0:120),XCOORD(0:120),YCOORD(0:120),TLOAD,TDIST,
     *         P1,P2,P3,FMAXLD,FMINLD,FMAXLN,FMINLN,DEPOT4,TR(3),
     *         TRY(120,120),TWTLM
      DIMENSION SORT(6000),LOKK(120)
      DIMENSION TRADE(7,500)
      COMMON IROUTE,NCITY,IRUN,START,END,WTLIM,DISTLM,ALLOW,NPERM,
     *ICOUNT,LDDVLM,LNDVLM,DLIMIT,MAXLD,MINLD,MAXLN,MINLN,D1,D2,D3,D4,
     *D5,D6,DEPOT,PRED,SUCC,TRUCK,DEMAND,LENGTH,LOAD,CUMLD,CUMLN,TEMTRK,
     *FEASLD,FEASLN,PERMPR,PERMSU,PERMTR,ISORT,JSORT,SORT,DIST,XCOORD,
     *YCOORD,LOKK,TRADE,NTRADE,DSEED
C
C
C
C
      NTRADE=0
      MODE='F'
      TLOAD=0
      TDIST=0
      START=GGUBFS(DSEED)*NCITY+1
      END=PRED(START)
      DO 1 I=1,IROUTE
        FEASLD(I)=LOAD(I)
        FEASLN(I)=LENGTH(I)
        TDIST=TDIST+LENGTH(I)
    1   TLOAD=TLOAD+LOAD(I)
      DO 2 I=1,NCITY+IROUTE
        FPRED(I)=PRED(I)
    2   FSUCC(I)=SUCC(I)
      DO 3 I=1,NCITY+IROUTE
      DO 3 J=1,NCITY+IROUTE
    3   TRY(I,J)=0
C
C
C
C     DETERMINE LOAD AND LENGTH DEVIATIONS
C
C
      LNDEV=MAXLN-MINLN
      LDDEV=MAXLD-MINLD
C
C
C
C     BEGIN ITERATIONS
C
C
      TWTLM=MIN0(WTLIM,MAXLD+LDDEV+LDRLX)
      P1=PRED(START)
    6 P1=SUCC(P1)
      IF(P1.EQ.PRED(PRED(END))) THEN
        RETURN
      ENDIF
      IF(LOKK(P1).EQ.1) GOTO 6
      P2=P1
    7 P2=SUCC(P2)
      IF(P2.EQ.PRED(END)) GOTO 6
      IF(LOKK(P2).EQ.1) GOTO 7
      P3=P2
    8 P3=SUCC(P3)
      IF(P3.EQ.END) GOTO 7
      IF(LOKK(P3).EQ.1) GOTO 8
C
C
C
C     IF ALL ARCS ARE IN THE SAME ROUTE, IGNORE THE EXCHANGE
C
C
      IF(TRUCK(P1).EQ.TRUCK(P2).AND.TRUCK(P1).EQ.TRUCK(P3)) GOTO 8
C
```

```
C
C
C     DETERMINE WHETHER AT LEAST ONE OF THE ARCS IS IN THE HEAVY ROUTE
C     OR IN THE LIGHT ROUTE.  IF NOT, IGNORE THE EXCHANGE.
C
C
      IF(LOAD(TRUCK(P1)).LE.O.OR.LOAD(TRUCK(P2)).LE.O.OR.
     *   LOAD(TRUCK(P3)).LE.O) GOTO 8
      IF(LOAD(TRUCK(P1)).EQ.MAXLD.OR.LOAD(TRUCK(P1)).EQ.MINLD)
     *   GOTO 99
      IF(LOAD(TRUCK(P2)).EQ.MAXLD.OR.LOAD(TRUCK(P2)).EQ.MINLD)
     *   GOTO 99
      IF(LOAD(TRUCK(P3)).EQ.MAXLD.OR.LOAD(TRUCK(P3)).EQ.MINLD)
     *   GOTO 99
C     ELSE
      GOTO 8
C
C
   99 CONTINUE
      IF(LDRLX.GT.O) THEN
        IF(P1.GT.NCITY) GOTO 8
        IF(P2.GT.NCITY) GOTO 8
        IF(P3.GT.NCITY) GOTO 8
      ENDIF
      IF(TRUCK(P1).NE.TRUCK(P2).AND.TRUCK(P1).NE.TRUCK(P3).AND.
     *   TRUCK(P3).NE.TRUCK(P2)) THEN
        LONER=0
        GOTO 9
      ENDIF
      IF(TRUCK(P2).EQ.TRUCK(P3)) THEN
        LONER=1
        GOTO 11
      ENDIF
      IF(TRUCK(P1).EQ.TRUCK(P3)) THEN
        LONER=2
        GOTO 11
      ENDIF
      IF(TRUCK(P1).EQ.TRUCK(P2)) THEN
        LONER=3
        GOTO 11
      ENDIF
C
C
C
C     TYPE I EXCHANGE - LOAD FEASIBILITY TEST
C
C
    9 NTYPE=1
        IF(CUMLD(P1)+CUMLD(P2).GT.TWTLM) GOTO 10
        IF(LOAD(TRUCK(P1))-CUMLD(P1)+CUMLD(P3).GT.TWTLM) GOTO 10
        IF(LOAD(TRUCK(P2))-CUMLD(P2)+LOAD(TRUCK(P3))-CUMLD(P3).GT.TWTLM)
     *    GOTO 10
        FEASLD(TRUCK(P1))=CUMLD(P1)+CUMLD(P2)
        FEASLD(TRUCK(P2))=LOAD(TRUCK(P1))-CUMLD(P1)+CUMLD(P3)
        FEASLD(TRUCK(P3))=LOAD(TRUCK(P2))-CUMLD(P2)+LOAD(TRUCK(P3))
     *                    -CUMLD(P3)
        NUMTRK=3
        TR(1)=P1
        TR(2)=P2
        TR(3)=P3
        GOTO 50
C
C
C
C     TYPE II EXCHANGE - LOAD FEASIBILITY TEST
C
C
   10 NTYPE=2
        IF(CUMLD(P1)+LOAD(TRUCK(P2))-CUMLD(P2).GT.TWTLM) GOTO 11
        IF(CUMLD(P2)+CUMLD(P3).GT.TWTLM) GOTO 11
        IF(LOAD(TRUCK(P1))-CUMLD(P1)+LOAD(TRUCK(P3))-CUMLD(P3).GT.TWTLM)
```

```
     *    GOTO 11
          FEASLD(TRUCK(P1))=CUMLD(P1)+LOAD(TRUCK(P2))-CUMLD(P2)
          FEASLD(TRUCK(P2))=CUMLD(P2)+CUMLD(P3)
          FEASLD(TRUCK(P3))=LOAD(TRUCK(P1))-CUMLD(P1)+LOAD(TRUCK(P3))
     *                     -CUMLD(P3)
          NUMTRK=3
          TR(1)=P1
          TR(2)=P2
          TR(3)=P3
          GOTO 50
C
C
C     TYPE III EXCHANGE - LOAD FEASIBILITY TEST
C
C
   11 NTYPE=3
        IF(LONER.EQ.0) THEN
          IF(P1.GT.NCITY.AND.P2.GT.NCITY.AND.P3.GT.NCITY) GOTO 12
          IF(CUMLD(P1)+LOAD(TRUCK(P2))-CUMLD(P2).GT.TWTLM) GOTO 12
          IF(LOAD(TRUCK(P1))-CUMLD(P1)+CUMLD(P3).GT.TWTLM) GOTO 12
          IF(CUMLD(P2)+LOAD(TRUCK(P3))-CUMLD(P3).GT.TWTLM) GOTO 12
          FEASLD(TRUCK(P1))=CUMLD(P1)+LOAD(TRUCK(P2))-CUMLD(P2)
          FEASLD(TRUCK(P2))=LOAD(TRUCK(P1))-CUMLD(P1)+CUMLD(P3)
          FEASLD(TRUCK(P3))=CUMLD(P2)+LOAD(TRUCK(P3))-CUMLD(P3)
          NUMTRK=3
          TR(1)=P1
          TR(2)=P2
          TR(3)=P3
          GOTO 50
        ENDIF
        IF(LONER.EQ.1) THEN
          IF(LOAD(TRUCK(P1))+CUMLD(P3)-CUMLD(P2).GT.TWTLM) GOTO 13
          FEASLD(TRUCK(P1))=LOAD(TRUCK(P1))+CUMLD(P3)-CUMLD(P2)
          FEASLD(TRUCK(P2))=LOAD(TRUCK(P2))-CUMLD(P3)+CUMLD(P2)
          NUMTRK=2
          TR(1)=P1
          TR(2)=P2
          GOTO 50
        ENDIF
        IF(LONER.EQ.3) THEN
          IF(LOAD(TRUCK(P3))+CUMLD(P2)-CUMLD(P1).GT.TWTLM) GOTO 14
          FEASLD(TRUCK(P2))=LOAD(TRUCK(P2))-CUMLD(P2)+CUMLD(P1)
          FEASLD(TRUCK(P3))=LOAD(TRUCK(P3))+CUMLD(P2)-CUMLD(P1)
          NUMTRK=2
          TR(1)=P2
          TR(2)=P3
          GOTO 50
        ENDIF
        IF(LONER.EQ.2) THEN
          IF(LOAD(TRUCK(P2))+CUMLD(P1)-CUMLD(P3).GT.TWTLM) GOTO 13
          FEASLD(TRUCK(P2))=LOAD(TRUCK(P2))+CUMLD(P1)-CUMLD(P3)
          FEASLD(TRUCK(P1))=LOAD(TRUCK(P1))-CUMLD(P1)+CUMLD(P3)
          NUMTRK=2
          TR(1)=P2
          TR(2)=P1
          GOTO 50
        ENDIF
C
C
C
C     TYPE IV EXCHANGE - LOAD FEASIBILITY TEST
C
C
   12 NTYPE=4
          IF(CUMLD(P1)+CUMLD(P3).GT.TWTLM) GOTO 13
          IF(LOAD(TRUCK(P1))-CUMLD(P1)+LOAD(TRUCK(P2))-CUMLD(P2).GT.TWTLM)
     *    GOTO 13
          IF(CUMLD(P2)+LOAD(TRUCK(P3))-CUMLD(P3).GT.TWTLM) GOTO 13
          FEASLD(TRUCK(P1))=CUMLD(P1)+CUMLD(P3)
          FEASLD(TRUCK(P2))=LOAD(TRUCK(P1))-CUMLD(P1)+LOAD(TRUCK(P2))
     *                     -CUMLD(P2)
```

```
        FEASLD(TRUCK(P3))=CUMLD(P2)+LOAD(TRUCK(P3))-CUMLD(P3)
        NUMTRK=3
        TR(1)=P1
        TR(2)=P2
        TR(3)=P3
        GOTO 50
C
C
C
C     TYPE V EXCHANGE - LOAD FEASIBILITY TEST
C
C
   13 NTYPE=5
      IF(TRY(P1,P2).EQ.1) GOTO 14
      TRY(P1,P2)=1
      IF(LONER.EQ.3) GOTO 14
      IF(CUMLD(P1)+CUMLD(P2).GT.TWTLM) GOTO 14
      IF(LOAD(TRUCK(P1))-CUMLD(P1)+LOAD(TRUCK(P2))-CUMLD(P2).GT.TWTLM)
     *  GOTO 14
      FEASLD(TRUCK(P1))=CUMLD(P1)+CUMLD(P2)
      FEASLD(TRUCK(P2))=LOAD(TRUCK(P1))-CUMLD(P1)+LOAD(TRUCK(P2))
     *                  -CUMLD(P2)
      NUMTRK=2
      TR(1)=P1
      TR(2)=P2
      GOTO 50
C
C
C
C     TYPE VI EXCHANGE - LOAD FEASIBILITY TEST
C
C
   14 NTYPE=6
      IF(TRY(P2,P3).EQ.1) GOTO 15
      TRY(P2,P3)=1
      IF(LONER.EQ.1) GOTO 15
      IF(CUMLD(P2)+CUMLD(P3).GT.TWTLM) GOTO 15
      IF(LOAD(TRUCK(P2))-CUMLD(P2)+LOAD(TRUCK(P3))-CUMLD(P3).GT.TWTLM)
     *  GOTO 15
      FEASLD(TRUCK(P2))=CUMLD(P2)+CUMLD(P3)
      FEASLD(TRUCK(P3))=LOAD(TRUCK(P2))-CUMLD(P2)+LOAD(TRUCK(P3))
     *                  -CUMLD(P3)
      NUMTRK=2
      TR(1)=P2
      TR(2)=P3
      GOTO 50
C
C
C
C     TYPE VII EXCHANGE - LOAD FEASIBILITY TEST
C
C
   15 NTYPE=7
      IF(TRY(P1,P3).EQ.1) GOTO 8
      TRY(P1,P3)=1
      IF(LONER.EQ.2) GOTO 8
      IF(CUMLD(P1)+CUMLD(P3).GT.TWTLM) GOTO 8
      IF(LOAD(TRUCK(P1))-CUMLD(P1)+LOAD(TRUCK(P3))-CUMLD(P3).GT.TWTLM)
     *  GOTO 8
      FEASLD(TRUCK(P1))=CUMLD(P1)+CUMLD(P3)
      FEASLD(TRUCK(P3))=LOAD(TRUCK(P1))-CUMLD(P1)+LOAD(TRUCK(P3))
     *                  -CUMLD(P3)
      NUMTRK=2
      TR(1)=P1
      TR(2)=P3
      GOTO 50
C
C
C
C     DETERMINE WHETHER MAXIMUM LOAD DEVIATION HAS BEEN REDUCED.
C
```

```
C
   50 FMAXLD=-99
      FMINLD=999999
      DO 51 I=1,IROUTE
        IF(LOKK(DEPOT(I)).NE.O) GOTO 51
        IF(FEASLD(I).GT.FMAXLD) FMAXLD=FEASLD(I)
        IF(FEASLD(I).LT.FMINLD.AND.FEASLD(I).NE.O) FMINLD=FEASLD(I)
   51 CONTINUE
      IF(FMAXLD-FMINLD.GE.LDDEV+LDRLX) THEN
        FEASLD(TRUCK(P1))=LOAD(TRUCK(P1))
        FEASLD(TRUCK(P2))=LOAD(TRUCK(P2))
        FEASLD(TRUCK(P3))=LOAD(TRUCK(P3))
        IF(LONER.EQ.O) GOTO (10,11,12,13,14,15,8), NTYPE
        IF(LONER.EQ.1) GOTO (11,11,13,13,15,15,8), NTYPE
        IF(LONER.EQ.2) GOTO (11,11,13,13,14,8,8), NTYPE
        IF(LONER.EQ.3) GOTO (11,11,14,14,14,15,8), NTYPE
      ENDIF
C
C
C
C     ESTABLISH POTENTIAL ROUTE STRUCTURE RESULTING FROM EXCHANGE
C
C
      IF(NTYPE.LE.4) CALL FXCH3(P1,P2,P3,FPRED,FSUCC,NTYPE)
      IF(NTYPE.EQ.5) CALL FXCH2(P1,P2,FPRED,FSUCC,NULL)
      IF(NTYPE.EQ.6) CALL FXCH2(P2,P3,FPRED,FSUCC,NULL)
      IF(NTYPE.EQ.7) CALL FXCH2(P1,P3,FPRED,FSUCC,NULL)
      IF(NTYPE.GT.4.AND.NULL.EQ.1) THEN
        DO 49 I=1,NCITY+IROUTE
          FPRED(I)=PRED(I)
   49     FSUCC(I)=SUCC(I)
        FEASLD(TRUCK(P1))=LOAD(TRUCK(P1))
        FEASLD(TRUCK(P2))=LOAD(TRUCK(P2))
        FEASLD(TRUCK(P3))=LOAD(TRUCK(P3))
        IF (LONER.EQ.O) GOTO (10,11,12,13,14,15,8), NTYPE
        IF(LONER.EQ.1) GOTO (11,11,13,13,15,15,8), NTYPE
        IF(LONER.EQ.2) GOTO (11,11,13,13,14,8,8), NTYPE
        IF(LONER.EQ.3) GOTO (11,11,14,14,14,15,8),NTYPE
      ENDIF
C
C
C
C     SOLVE TSP FOR EACH ROUTE AFFECTED BY THE EXCHANGE
C
C
      DO 55 I=1,NUMTRK
        ISTRT=DEPOT(TRUCK(TR(I)))
        NEXT=FSUCC(ISTRT)
   52   NODE=NEXT
        IF(NODE.GT.NCITY) THEN
          IEND=FPRED(NODE)
          CALL TSP(ISTRT,IEND,FPRED,FSUCC,LNGTH,DIST,ALLOW)
          IF(LNGTH.GT.DISTLM) GOTO 63
          FEASLN(TRUCK(TR(I)))=LNGTH
          GOTO 55
        ENDIF
        NEXT=FSUCC(NODE)
        GOTO 52
   55 CONTINUE
C
C
C
C     DETERMINE TOTAL DISTANCE INCREASE.
C
C
      LTDIST=O
      DO 60 I=1,IROUTE
   60 LTDIST=LTDIST+FEASLN(I)
C
C
C
```

```
C     DETERMINE ROUTE LENGTH DEVIATION.
C
C
      FMAXLN=-99
      FMINLN=999999
      DO 61 I=1,IROUTE
        IF(LOKK(DEPOT(I)).NE.O) GOTO 61
        IF(FEASLN(I).GT.FMAXLN) FMAXLN=FEASLN(I)
        IF(FEASLN(I).LT.FMINLN.AND.FEASLN(I).NE.O) FMINLN=FEASLN(I)
   61 CONTINUE
C
C
C
C     TRADEOFF ANALYSIS.
C
C
      IF(LDRLX.GT.O) GOTO 62
C     ELSE
      IF(LTDIST.GT.DLIMIT.OR.FMAXLN-FMINLN.GT.LNDVLM) THEN
        NTRADE=NTRADE+1
        IF(NTRADE.GT.500) NTRADE=500
        TRADE(1,NTRADE)=LDDEV-(FMAXLD-FMINLD)
        TRADE(1,NTRADE)=-TRADE(1,NTRADE)
        TRADE(2,NTRADE)=LTDIST-TDIST
        TRADE(3,NTRADE)=FMAXLN-FMINLN-LNDEV
        TRADE(4,NTRADE)=P1
        TRADE(5,NTRADE)=P2
        TRADE(6,NTRADE)=P3
        TRADE(7,NTRADE)=NTYPE
      ENDIF
C
C
C
   62 IF(LTDIST.GT.DLIMIT.OR.FMAXLN-FMINLN.GT.LNDVLM) THEN
C
C
C      REBUILD THE ROUTE STRUCTURE.
C
   63  DO 64 I=NCITY+1,NCITY+IROUTE
         FSUCC(FPRED(I))=SUCC(FPRED(I))
   64    FPRED(I)=PRED(I)
       DO 66 I=1,NUMTRK
         FEASLD(TRUCK(TR(I)))=LOAD(TRUCK(TR(I)))
         FEASLN(TRUCK(TR(I)))=LENGTH(TRUCK(TR(I)))
         NODE=DEPOT(TRUCK(TR(I)))
         FPRED(NODE)=PRED(NODE)
         FSUCC(NODE)=SUCC(NODE)
         NEXT=SUCC(NODE)
   65    NODE=NEXT
         IF(NODE.GT.NCITY) GOTO 66
         FPRED(NODE)=PRED(NODE)
         FSUCC(NODE)=SUCC(NODE)
         NEXT=SUCC(NODE)
         GOTO 65
   66  CONTINUE
       IF(LONER.EQ.O) GOTO (10,11,12,13,14,15,8), NTYPE
       IF(LONER.EQ.1) GOTO (11,11,13,13,15,15,8), NTYPE
       IF(LONER.EQ.2) GOTO (11,11,13,13,14,8,8),  NTYPE
       IF(LONER.EQ.3) GOTO (11,11,14,14,14,15,8), NTYPE
       CONTINUE
      ENDIF
C
C
C
C
C     CHANGE ROUTE STRUCTURE
C
C
      MAXLN=FMAXLN
      MINLN=FMINLN
      MAXLD=FMAXLD
```

```
      MINLD=FMINLD
      LNDEV=MAXLN-MINLN
      LDDEV=MAXLD-MINLD
      DO 67 I=1,NCITY+IROUTE
        PRED(I)=FPRED(I)
   67   SUCC(I)=FSUCC(I)
      NEXT=NCITY+1
      INSIDE=0
   68 NODE=NEXT
      IF(NODE.EQ.NCITY+1.AND.INSIDE.EQ.1) GOTO 69
      INSIDE=1
      IF(NODE.GT.NCITY) THEN
        ITRK=TRUCK(NODE)
        ILD=0
        ILN=0
      ENDIF
      ILD=ILD+DEMAND(NODE)
      IF(NODE.LE.NCITY) ILN=ILN+DIST(NODE,PRED(NODE))+ALLOW
      CUMLD(NODE)=ILD
      CUMLN(NODE)=ILN
      TRUCK(NODE)=ITRK
      FEASLD(ITRK)=ILD
      NEXT=SUCC(NODE)
      IF(NEXT.GT.NCITY) FEASLN(ITRK)=ILN+DIST(NODE,NEXT)
      GOTO 68
   69 TDIST=0
      TLOAD=0
      DO 70 I=1,IROUTE
        LOAD(I)=FEASLD(I)
        LENGTH(I)=FEASLN(I)
        TLOAD=TLOAD+LOAD(I)
   70   TDIST=TDIST+LENGTH(I)
C
C
C
C     ROTATE
C
C
      TWTLM=MIN0(WTLIM,MAXLD+LDDEV)
      IRLX=LDRLX
      LDRLX=0
      NTRADE=0
      ISTART=END
      IF(IRLX.GT.0) ISTART=DEPOT(TRUCK(PRED(ISTART)))
      END=PRED(ISTART)
      START=ISTART
      P1=END
      DO 71 I=1,NCITY+IROUTE
      DO 71 J=1,NCITY+IROUTE
   71 TRY(I,J)=0
      GOTO 6
      END
C
C
C***********************************************************************
C
      SUBROUTINE NONDOM(ITRADE,NT)
C
C
C     THIS SUBROUTINE ELIMINATES ALL DOMINATED TRADEOFFS FROM THE SET
C     OF TRADEOFFS.
C
C***********************************************************************
C
      DIMENSION ITRADE(7,500)
C
C
      IF(NT.LE.1) RETURN
      N=1
      DO 10 I=2,NT
        N1=N
```

```
        DO 9 K=1,N1
          IF(ITRADE(2,I).GE.ITRADE(2,K).AND.ITRADE(3,I).GE.ITRADE(3,K))
     *      GOTO 10
    9   CONTINUE
        N=N+1
        ITRADE(1,N)=ITRADE(1,I)
        ITRADE(2,N)=ITRADE(2,I)
        ITRADE(3,N)=ITRADE(3,I)
        ITRADE(4,N)=ITRADE(4,I)
        ITRADE(5,N)=ITRADE(5,I)
        ITRADE(6,N)=ITRADE(6,I)
        ITRADE(7,N)=ITRADE(7,I)
   10 CONTINUE
      NT=N
      RETURN
      END
C
C
C*********************************************************************
C
      SUBROUTINE DISPLA(NUMBER,IROUTE,NCITY,PNAME,XCOORD,YCOORD,TIME)
C
C
C     THIS SUBROUTINE IS USED TO DISPLAY A PRIOR SOLUTION TO
C     THE WORKLOAD-BALANCED VEHICLE ROUTING PROBLEM.
C
C*********************************************************************
      CHARACTER*16 OBJ(12)
      CHARACTER*44 PNAME
      INTEGER PRED(12,120),SUCC(12,120),TRUCK(12,120),CUMLN(12,120),
     *CUMLD(12,120),DEPOT(12,20),LOAD(12,20),LENGTH(12,20),MAXLN(12),
     *MINLN(12),MAXLD(12),MINLD(12),ITRADE(12,7,500),LDDVLM(12),
     *LNDVLM(12),DLIMIT(12),NT(12),RTSIZE(20),XCOORD(0:120),
     *YCOORD(0:120),LOKK(12,120),TDIST
      DIMENSION XCENTR(20),YCENTR(20)
C
      COMMON /STORE/ PRED,SUCC,TRUCK,CUMLN,CUMLD,DEPOT,LOAD,LENGTH,
     *MAXLN,MINLN,MAXLD,MINLD,ITRADE,NT,LDDVLM,LNDVLM,DLIMIT,LOKK,OBJ
C
C
C
C     CALCULATE TOTAL DISTANCE, LOAD DEVIATION, AND LENGTH DEVIATION.
C
C
      TDIST=0
      DO 1 I=1,IROUTE
    1   TDIST=TDIST+LENGTH(NUMBER,I)
      LDDEV=MAXLD(NUMBER)-MINLD(NUMBER)
      LNDEV=MAXLN(NUMBER)-MINLN(NUMBER)
C
C
C
C     DISPLAY ROUTES ON GRAPHICS SCREEN
C
C
      CALL NEWPAG
      CALL TWINDO(0,4095,0,3120)
      CALL MOVABS(0,0)
      CALL DRWABS(4095,0)
      CALL DRWABS(4095,3120)
      CALL DRWABS(0,3120)
      CALL DRWABS(0,0)
      CALL MOVABS(1030,0)
      CALL DRWABS(1030,3120)
      CALL MOVABS(1030,2800)
      CALL DRWABS(4095,2800)
      CALL MOVABS(1130,2950)
      CALL AOUTST(44,PNAME)
      CALL TWINDO(1030,4095,0,2800)
      DO 2 J=1,IROUTE
        RTSIZE(J)=0
```

```
        XCENTR(J)=0.0
      2   YCENTR(J)=0.0
        NEXT=NCITY+1
        X=XCOORD(NEXT)
        Y=YCOORD(NEXT)
        XCENTR(TRUCK(NUMBER,NEXT))=XCENTR(TRUCK(NUMBER,NEXT))+X
        YCENTR(TRUCK(NUMBER,NEXT))=YCENTR(TRUCK(NUMBER,NEXT))+Y
        RTSIZE(TRUCK(NUMBER,NEXT))=RTSIZE(TRUCK(NUMBER,NEXT))+1
        CALL MOVEA(X,Y)
        CALL MOVREL(40,0)
        CALL DRWREL(0,40)
        CALL DRWREL(-80,0)
        CALL DRWREL(0,-80)
        CALL DRWREL(80,0)
        CALL DRWREL(0,40)
        CALL MOVEA(X,Y)
        INSIDE=0
      3 NODE=NEXT
        IF(NODE.EQ.NCITY+1.AND.INSIDE.EQ.1) GOTO 4
        INSIDE=1
        X=XCOORD(NODE)
        Y=YCOORD(NODE)
        XCENTR(TRUCK(NUMBER,NODE))=XCENTR(TRUCK(NUMBER,NODE))+X
        YCENTR(TRUCK(NUMBER,NODE))=YCENTR(TRUCK(NUMBER,NODE))+Y
        RTSIZE(TRUCK(NUMBER,NODE))=RTSIZE(TRUCK(NUMBER,NODE))+1
        CALL DRAWA(X,Y)
        ICHR1=NODE/100
        ICHR2=NODE/10 - ICHR1*10
        ICHR3=NODE - (ICHR1*100+ICHR2*10)
        IF(NODE.LE.NCITY) THEN
          CALL MOVREL(20,0)
          CALL DRWREL(0,20)
          CALL DRWREL(-40,0)
          CALL DRWREL(0,-40)
          CALL DRWREL(40,0)
          CALL DRWREL(0,20)
        ENDIF
        CALL MOVEA(X,Y)
        NEXT=SUCC(NUMBER,NODE)
        GOTO 3
      4 CONTINUE
        X=XCOORD(NODE)
        Y=YCOORD(NODE)
        CALL DRAWA(X,Y)
        DO 5 J=1,IROUTE
          IF(LOAD(NUMBER,J).GT.0) THEN
          XCENTR(J)=XCENTR(J)/FLOAT(RTSIZE(J))
          YCENTR(J)=YCENTR(J)/FLOAT(RTSIZE(J))
          CALL MOVEA(XCENTR(J),YCENTR(J))
          ICHR1=J/10
          ICHR2=J-ICHR1*10
          IF(ICHR1.NE.0) CALL ANCHO(ICHR1+48)
          CALL ANCHO(ICHR2+48)
          ENDIF
      5 CONTINUE
C
C
C
C       DISPLAY SOLUTION STATISTICS
C
C
        CALL HOME
        CALL ANMODE
        WRITE(6,100) NUMBER,OBJ(NUMBER)
    100 FORMAT(1H //1X,'SOLUTION NUMBER',I2//1X,A/' MINIMIZATION PROBLEM'
       *//' ROUTE  LOAD  LENGTH'/)
        DO 6 II=1,IROUTE
        IF(LOAD(NUMBER,IROUTE).LT.0) GOTO 5
        WRITE(6,101) II,LOAD(NUMBER,II),LENGTH(NUMBER,II)
      6 CONTINUE
    101 FORMAT(I4,I8,1X,I5)
```

```
      WRITE(6,102) TDIST,LDDEV,LNDEV
  102 FORMAT(' TOT. DIST =',I6/' LOAD DEV. = ',I5/' LENGTH DEV. =',I4)
      CALL ELAPSE(ITIME)
      TIME=TIME+FLOAT(ITIME)/1000.
      WRITE(6,103) TIME
  103 FORMAT(1H ,'CPU SECONDS:',F6.2)
      WRITE(6,104)
  104 FORMAT(1H ,///////////,' HIT <RTN> TO CONTINUE')
      CALL TINPUT(MMMM)
      RETURN
      END
C
C
C
C*********************************************************************
C
      SUBROUTINE BKTRAK(NUMBER,ITRADE,NT,OBJ,OLDOBJ,LIMIT1,LIMIT2,
     *                  CNSTR1,CNSTR2)
C
C
C     THIS SUBROUTINE IS USED TO BACKTRACK TO A PRIOR SOLUTION OF THE
C     WORKLOAD-BALANCED VEHICLE ROUTING PROBLEM.
C
C*********************************************************************
      CHARACTER*44 PNAME,IPLACE
      CHARACTER*16 OBJ,STOBJ(12),CNSTR1,CNSTR2
      CHARACTER*1 ANSWER
      INTEGER EUCLID,CITY,XCOORD(0:120),YCOORD(0:120),DEMAND(0:120)
      INTEGER HEAD(120),TAIL(120),PRED(120),SUCC(120),ROUTES,TWGT,WTLIM
      INTEGER DIST,ALLOW,TDIST,DISTLM,TRUCK,IR(100),IFLAG(40),
     * PERMI(40),PERMJ(40)
      DOUBLE PRECISION DSEED
      INTEGER START,END,POINT1,POINT2,D,FEASLD(20),FEASLN(20)
      INTEGER FTRUCK,FWD,BACK,TEMTRK(120),CUMLD(120),CUMLN(120)
      INTEGER FOUND,TRCNT,TRK,TAG,D1,D2,D3,D4,D5,D6,D7,D8
      INTEGER FSTART,FEND,FPRED(120),FSUCC(120),BACTRT,DSTRLX
      INTEGER PERMPR(120),PERMSU(120),PERMTR(120),RTSIZE(20)
      INTEGER BESTRT,BESTDS,BCUMLN(120),BCUMLD(120),BESTP(120)
      INTEGER BESTS(120),BESTLN(20),BESTLD(20),BDEPOT(20),BESTTR(120)
      INTEGER DLIMIT,TARRAY(40),DEPOT(20),CLRNDX(12),BLDDEV,BLNDEV
      INTEGER BMAXLD,BMINLD,BMAXLN,BMINLN,LOKK(120),STLOKK(12,120)
      INTEGER BNTRAD,ITRADE(7,500),OLDOBJ
      INTEGER SOLNO,STPRED(12,120),STSUCC(12,120),STTRK(12,120),
     *STCULN(12,120),STCULD(12,120),STDEP(12,20),STLOAD(12,20),
     *STLNTH(12,20),STMXLN(12),STMNLN(12),STMXLD(12),STMNLD(12),
     *STITRD(12,7,500),STLDVL(12),STLNVL(12),STDLIM(12),STNT(12)
      DIMENSION BTRADE(7,500),TRADE(7,500),WK(14)
      DIMENSION DIST(0:120,0:120),SAVING(3,6000),SORT(6000),PERMSV(40)
      DIMENSION ISORT(6000),JSORT(6000),LOAD(120),TRUCK(120)
      DIMENSION LENGTH(120),WORK(6),XCENTR(20),YCENTR(20)
      COMMON IROUTE,NCITY,IRUN,START,END,WTLIM,DISTLM,ALLOW,NPERM,
     *ICOUNT,LDDVLM,LNDVLM,DLIMIT,MAXLD,MINLD,MAXLN,MINLN,D1,D2,D3,D4,
     *D5,D6,DEPOT,PRED,SUCC,TRUCK,DEMAND,LENGTH,LOAD,CUMLD,CUMLN,TEMTRK,
     *FEASLD,FEASLN,PERMPR,PERMSU,PERMTR,ISORT,JSORT,SORT,DIST,XCOORD,
     *YCOORD,LOKK,TRADE,NTRADE,DSEED
      COMMON /STORE/STPRED,STSUCC,STTRK,STCULN,STCULD,STDEP,STLOAD,
     *STLNTH,STMXLN,STMNLN,STMXLD,STMNLD,STITRD,STNT,STLDVL,STLNVL,
     *STDLIM,STLOKK,STOBJ
C
C
C
C     RESET ALL PROBLEM PARAMETERS TO PRIOR VALUES.
C
C
      DO 1 I=1,NCITY+IROUTE
        PRED(I)=STPRED(NUMBER,I)
        SUCC(I)=STSUCC(NUMBER,I)
        LOKK(I)=STLOKK(NUMBER,I)
        TRUCK(I)=STTRK(NUMBER,I)
        CUMLN(I)=STCULN(NUMBER,I)
    1   CUMLD(I)=STCULD(NUMBER,I)
```

```
      DO 2 I=1,IROUTE
        DEPOT(I)=STDEP(NUMBER,I)
        LOAD(I)=STLOAD(NUMBER,I)
    2   LENGTH(I)=STLNTH(NUMBER,I)
      MAXLN=STMXLN(NUMBER)
      MINLN=STMNLN(NUMBER)
      MAXLD=STMXLD(NUMBER)
      MINLD=STMNLD(NUMBER)
      NT=STNT(NUMBER)
      DO 3 I=1,NT
      DO 3 J=1,7
    3   ITRADE(J,I)=STITRD(NUMBER,J,I)
      LDDVLM=STLDVL(NUMBER)
      LNDVLM=STLNVL(NUMBER)
      DLIMIT=STDLIM(NUMBER)
      OBJ=STOBJ(NUMBER)
      IF(OBJ.EQ.'TOTAL DISTANCE') THEN
        LIMIT1=LDDVLM
        LIMIT2=LNDVLM
        CNSTR1='LOAD DEVIATION'
        CNSTR2='LENGTH DEVIATION'
        OLDOBJ=1
        RETURN
      ENDIF
      IF(OBJ.EQ.'LOAD DEVIATION') THEN
        LIMIT1=DLIMIT
        LIMIT2=LNDVLM
        CNSTR1='TOTAL DISTANCE'
        CNSTR2='LENGTH DEVIATION'
        OLDOBJ=2
        RETURN
      ENDIF
      IF(OBJ.EQ.'LENGTH DEVIATION') THEN
        LIMIT1=DLIMIT
        LIMIT2=LDDVLM
        CNSTR1='TOTAL DISTANCE'
        CNSTR2='LOAD DEVIATION'
        OLDOBJ=3
        RETURN
      ENDIF
      END
C
C
C
C*********************************************************************
C
      SUBROUTINE ADJUST(NUMBER)
C
C
C     THIS SUBROUTINE ACCEPTS MANUAL ADJUSTMENTS TO A VEHICLE ROUTE AND
C     EVALUATES THE IMPACT OF THOSE CHANGES ON THE LENGTH OF THE ALTERED
C     ROUTE.
C
C*********************************************************************
C
C
      DOUBLE PRECISION DSEED
      INTEGER START,END,WTLIM,DISTLM,ALLOW,DLIMIT,D1,D2,D3,D4,D5,D6,
     *DEPOT(20),PRED(120),SUCC(120),TRUCK(120),DEMAND(0:120),LENGTH(120)
     *,LOAD(120),CUMLD(120),CUMLN(120),TEMTRK(120),FEASLD(20),FEASLN(20)
     *,PERMPR(120),PERMSU(120),PERMTR(120),ISORT(6000),JSORT(6000),
     *DIST(0:120,0:120),XCOORD(0:120),YCOORD(0:120),LOKK(120)
      INTEGER XSTART,XEND,XPRED(120),XSUCC(120),XCUMLD(120),XCUMLN(120),
     *RTSIZE(20)
      DIMENSION TRADE(7,500),XCENTR(20),YCENTR(20),SORT(6000)
      COMMON IROUTE,NCITY,IRUN,START,END,WTLIM,DISTLM,ALLOW,NPERM,
     *ICOUNT,LDDVLM,LNDVLM,DLIMIT,MAXLD,MINLD,MAXLN,MINLN,D1,D2,D3,D4,
     *D5,D6,DEPOT,PRED,SUCC,TRUCK,DEMAND,LENGTH,LOAD,CUMLD,CUMLN,TEMTRK,
     *FEASLD,FEASLN,PERMPR,PERMSU,PERMTR,ISORT,JSORT,SORT,DIST,XCOORD,
     *YCOORD,LOKK,TRADE,NTRADE,DSEED
C
```

```
C
C     SET UP TEMPORARY VARIABLES AND ARRAYS.
C
C
      XSTART=DEPOT(NUMBER)
      DO 10 I=1,IROUTE
        IF(TRUCK(PRED(DEPOT(I))).EQ.NUMBER) XEND=DEPOT(I)
   10 CONTINUE
      DO 20 I=1,NCITY+IROUTE
        XPRED(I)=PRED(I)
        XSUCC(I)=SUCC(I)
        XCUMLD(I)=CUMLD(I)
   20   XCUMLN(I)=CUMLN(I)
C
C
C
C     DEFINE VIRTUAL WINDOW AND SCREEN WINDOW FOR ROUTE TO BE ADJUSTED.
C
C
      MAXX=-99999
      MINX=999999
      MAXY=-99999
      MINY=999999
      DO 30 I=1,NCITY+IROUTE
        IF(TRUCK(I).NE.NUMBER) GOTO 30
        IF(XCOORD(I).GT.MAXX) MAXX=XCOORD(I)
        IF(XCOORD(I).LT.MINX) MINX=XCOORD(I)
        IF(YCOORD(I).GT.MAXY) MAXY=YCOORD(I)
        IF(YCOORD(I).LT.MINY) MINY=YCOORD(I)
   30 CONTINUE
      LIM=MAXO(MAXX-MINX,MAXY-MINY)
      X1=MINX-10
      X2=X1+FLOAT(LIM)+20.
      Y1=MINY-10
      Y2=Y1+FLOAT(LIM)+20.
      CALL DWINDO(X1,X2,Y1,Y2)
      CALL TWINDO(1030,4095,0,2800)
C
C
C
C     DISPLAY ROUTE TO BE ADJUSTED.
C
C
      CALL NEWPAG
      NEXT=DEPOT(NUMBER)
      DO 40 J=1,IROUTE
        RTSIZE(J)=0
        XCENTR(J)=0.0
   40   YCENTR(J)=0.0
      X=XCOORD(NEXT)
      Y=YCOORD(NEXT)
      XCENTR(TRUCK(NEXT))=XCENTR(TRUCK(NEXT)) + X
      YCENTR(TRUCK(NEXT))=YCENTR(TRUCK(NEXT)) + Y
      RTSIZE(TRUCK(NEXT))=RTSIZE(TRUCK(NEXT)) + 1
      CALL MOVEA(X,Y)
      CALL MOVREL(40,0)
      CALL DRWREL(0,40)
      CALL DRWREL(-80,0)
      CALL DRWREL(0,-80)
      CALL DRWREL(80,0)
      CALL DRWREL(0,40)
      CALL MOVEA(X,Y)
      INSIDE=0
   41 NODE=NEXT
      IF(NODE.GE.NCITY+1.AND.INSIDE.EQ.1) GOTO 42
      INSIDE=1
      X=XCOORD(NODE)
      Y=YCOORD(NODE)
      XCENTR(TRUCK(NODE))=XCENTR(TRUCK(NODE)) + X
      YCENTR(TRUCK(NODE))=YCENTR(TRUCK(NODE)) + Y
      RTSIZE(TRUCK(NODE))=RTSIZE(TRUCK(NODE)) + 1
```

```
      CALL DRAWA(X,Y)
      ICHR1=NODE/100
      ICHR2=NODE/10 - ICHR1*10
      ICHR3=NODE - (ICHR1*100 + ICHR2*10)
      IF(NODE.LE.NCITY) THEN
        CALL MOVREL(20,0)
        CALL DRWREL(0,20)
        CALL DRWREL(-40,0)
        CALL DRWREL(0,-40)
        CALL DRWREL(40,0)
        CALL DRWREL(0,20)
        CALL ANCHO(ICHR1+48)
        CALL ANCHO(ICHR2+48)
        CALL ANCHO(ICHR3+48)
      ENDIF
      CALL MOVEA(X,Y)
      NEXT=SUCC(NODE)
      GOTO 41
   42 CONTINUE
      X=XCOORD(NODE)
      Y=YCOORD(NODE)
      CALL DRAWA(X,Y)
      DO 43 J=NUMBER,NUMBER
        IF(LOAD(J).GT.0) THEN
          XCENTR(J)=XCENTR(J)/FLOAT(RTSIZE(J))
          YCENTR(J)=YCENTR(J)/FLOAT(RTSIZE(J))
          CALL MOVEA(XCENTR(J),YCENTR(J))
          ICHR1=J/10
          ICHR2=J-ICHR1*10
          IF(ICHR1.NE.0) CALL ANCHO(ICHR1+48)
          CALL ANCHO(ICHR2+48)
        ENDIF
   43 CONTINUE
C
C
C
C     READ IN CHANGES TO THE ROUTE.
C
C
      CALL HOME
      CALL ANMODE
      NEXT=XSTART
      WRITE(6,100)
   44 NODE=NEXT
      READ(5,*) I
      IF(I.EQ.999) GOTO 50
C     ELSE
      XSUCC(NODE)=I
      XPRED(I)=NODE
      NEXT=I
      GOTO 44
   50 XSUCC(NODE)=XEND
      XPRED(XEND)=NODE
C
C
C     CALCULATE EFFECT OF CHANGES.
C
C
      ILD=0
      ILN=0
      NEXT=XSTART
   60 NODE=NEXT
      IF(NODE.EQ.XEND) GOTO 70
      ILD=ILD+DEMAND(NODE)
      IF(NODE.LE.NCITY) ILN=ILN+DIST(NODE,XPRED(NODE))+ALLOW
      XCUMLD(NODE)=ILD
      XCUMLN(NODE)=ILN
      NEXT=XSUCC(NODE)
      GOTO 60
   70 ILN=ILN+DIST(NODE,XPRED(NODE))
      IF(ILN.GE.LENGTH(NUMBER)) THEN
```

```
        CALL ANMODE
        WRITE(6,102) ILN,LENGTH(NUMBER)
        CALL TINPUT(MMMM)
        RETURN
      ENDIF
      IF(ILN.LT.LENGTH(NUMBER)) THEN
        CALL ANMODE
        WRITE(6,103) ILN,LENGTH(NUMBER)
        LENGTH(NUMBER)=ILN
        DO 71 I=1,NCITY+IROUTE
          PRED(I)=XPRED(I)
          SUCC(I)=XSUCC(I)
          CUMLD(I)=XCUMLD(I)
   71     CUMLN(I)=XCUMLN(I)
        MAXLN=-999999
        MINLN=999999
        DO 72 I=1,IROUTE
          IF(LOKK(DEPOT(I)).EQ.1) GOTO 72
          IF(LENGTH(I).LT.MINLN.AND.LENGTH(I).NE.0) MINLN=LENGTH(I)
          IF(LENGTH(I).GT.MAXLN) MAXLN=LENGTH(I)
   72   CONTINUE
        CALL TINPUT(MMMM)
        RETURN
      ENDIF
  100 FORMAT(1H ,'ENTER NODES, IN ORDER,'/' END INPUT WITH 999'/' ?')
  102 FORMAT(1H ,'NEW ROUTE LENGTH =',I4/' OLD ROUTE LENGTH =',I4/' NO I
     *MPROVEMENT -- CHANGES NOT IMPLEMENTED'/' HIT <RTN> TO CONTINUE')
  103 FORMAT(1H ,'NEW ROUTE LENGTH =',I4/' OLD ROUTE LENGTH =',I4/' IMPR
     *OVEMENT IN ROUTE LENGTH -- CHANGES IMPLEMENTED'/' HIT <RTN> TO CON
     *TINUE')
      END
C
C
C***********************************************************************
C
      SUBROUTINE LOCK  (NUMBER,LOKK,NCITY,TRUCK,IROUTE)
C
C
C     THIS SUBROUTINE 'LOCKS OUT' (I.E., MAKES UNAVAILABLE FOR
C     CALCULATIONS OF ROUTE-LOAD AND ROUTE-LENGTH DEVIATION) A VEHICLE
C     ROUTE.
C
C***********************************************************************
C
C
      INTEGER TRUCK(120),LOKK(120)
C
      DO 10 I=1,NCITY+IROUTE
        IF(TRUCK(I).EQ.NUMBER) LOKK(I)=1
   10 CONTINUE
      RETURN
      END
```

VITA

Jerry Denver Allison

Candidate for the Degree of

Doctor of Philosophy

Thesis: WORKLOAD BALANCING IN VEHICLE ROUTING PROBLEMS

Major Field: Industrial Engineering and Management

Biographical:

Personal Data: Born in Mineral Wells, Texas, February 12, 1944, the son of Joseph Earl and Desla Frances Allison. Married to Susan Carol Fisk on August 19, 1978.

Education: Graduated from Weatherford High School, Weatherford, Texas, in May, 1962; received Bachelor of Science Degree in Industrial Engineering from the University of Texas at Arlington in January, 1968; received Master of Engineering Degree in Industrial Engineering from Texas A and M University in May, 1970; completed requirements for the Doctor of Philosophy degree at Oklahoma State University in December, 1986.

Professional Experience: General Engineer, U.S. Army Materiel Command, Rock Island, Illinois, February, 1968 to September, 1970; Senior Project Engineer, Mason and Hanger, Amarillo, Texas, October, 1971 to August, 1981; Teaching and Research Associate, School of Industrial Engineering and Management, Oklahoma State University, September, 1981 to June, 1986.

Professional Organizations: Institute of Industrial Engineers, Alpha Pi Mu, The Institute of Management Sciences.