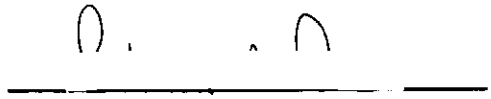
In presenting the dissertation as a partial fulfillment of the requirements for an advanced degree from the Georgia Institute of Technology, I agree that the Library of the Institute shall make it available for inspection and circulation in accordance with its regulations governing materials of this type. I agree that permission to copy from, or to publish from, this dissertation may be granted by the professor under whose direction it was written, or, in his absence, by the Dean of the Graduate Division when such copying or publication is solely for scholarly purposes and does not involve potential financial gain. It is understood that any copying from, or publication of, this dissertation which involves potential financial gain will not be allowed without written permission.



7/25/68

A TWO-PHASE ALGORITHM
FOR THE VEHICLE DELIVERY PROBLEM

A THESIS

Presented to

The Faculty of the Graduate Division

Studies and Research

by

Richard Andrew Dun

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in the School of Industrial and Systems Engineering

Georgia Institute of Technology

May 1971

A TWO-PHASE ALGORITHM

FOR THE VEHICLE DELIVERY PROBLEM

Approved:

Chairman

Date approved by Chairman: 5/27/71

ACKNOWLEDGMENTS

I wish to express my appreciation to Dr. V. E. Unger for his contributions to the work presented in this thesis and for serving as my advisor in both course work and thesis research.

I also wish to thank Dr. J. J. Jarvis and Dr. W. W. Hines for their helpful suggestions while serving on my reading committee.

Finally, I wish to thank my parents for their wisdom, guidance, and understanding and my wife for her uncommon patience and inspiration.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

SUMMARY

This thesis develops an algorithm for solving the vehicle delivery problem stated as follows: Consider m points each with a demand for deliveries, expressed in some convenient unit and denoted by q_i, and a terminal point with no demand. Let C be the capacity of the vehicles expressed in the same unit as demand and assume that

$$\max q_i < C < \sum_{i=1}^{n} q_i.$$

Further, assume that a symmetric distance matrix $D = [d_{ij}]$, which indicates the distance from any point i to the terminal and to any other point is known. Find the routing of the vehicles which will satisfy all demands without violating the capacity constraint on the vehicles while minimizing the total cost of delivery.

The algorithm is executed in two phases. Phase I generates a set of feasible routes based on vehicle capacity and then assigns a cost to each of these routes based on the solution to a travelling salesman problem. Phase II then uses the routes generated in Phase I to formulate the problem as a generalized set covering problem. The procedure for solving this formulation is a branch-and-bound process based on the solution of linear subproblems.

Computational results are obtained for a set of ten test problems, drawn from both the literature and actual delivery problems. Finally, desirable extensions of the algorithm are examined.

CHAPTER I

INTRODUCTION

Since 1956 a good deal of attention has been given to a problem which has been equivalently termed the vehicle delivery problem, the delivery problem, the dispatching problem, the truck dispatching problem, and the vehicle dispatching problem. The diverse considerations which the problem has received result from the fact that it arises in a variety of contexts and may exhibit any of a number of distinguishing characteristics. Basically, the problem concerns the transportation of products from one set of locations to another set of locations under certain restrictions which govern the nature of deliveries.

Generally, there are several characteristics of a product--volume, weight, length, etc.--which may affect the structure of the problem. Correspondingly, the vehicles may impose any of a number of restrictions on the problem depending on the number available and their capacities and operating characteristics. There may also arise differences in problem structure depending on whether deliveries are mandatory or optional, whether the quantities to be delivered are prespecified or to be selected, and whether these quantities must be delivered on a single visit or may be divided into several smaller quantities. Further, restrictions on the earliest or latest times for deliveries may appear. Finally, the objective in solving the problem may vary from minimizing time spent making deliveries, to minimizing the number of vehicles used, to

minimizing the total cost of delivery.

Literature Review

The earliest description of the vehicle delivery problem appears in a paper by Garvin et al. (1957). In an article on the applications of operations research in the oil refining industry, the authors discussed the problem of routing vehicles from a bulk terminal to individual service stations. Their problem involved only one product (i. e., one grade of gasoline) but did include consideration of vehicles with varying capacities. The formulation presented, which takes the form of a mixed-integer programming problem, had two unfortunate drawbacks: (1) at the time there was ano known method of solving an integer-restricted problem optimally and (2) the number of variables rapidly became unweildy as the number of stations increased. More recent developments in integer programming have solved the problem of obtaining optimal integer solutions for small problems; however, due to the number of variables involved, it is doubtful that any existing algorithm could be efficiently applied to their formulation.

Dantzig and Ramser (1959) discussed what they called the truck dispatching problem. Their definition is as follows: Consider N points each with a demand for deliveries of q_i and a terminal point with no demand. Let C be the capacity of the vehicles and assume that

$$\max q_i < C < \sum_{i=1}^{n} q_i .$$

Further, assume that a symmetric distance matrix $D = [d_{ij}]$, which indicates the distance from any point i to the terminal and to any

other point, is known. Find the routing for the vehicles that will satisfy the demands without violating the capacity constraint on the vehicles while minimizing the distance travelled.

The authors were unable to develop a model which would allow an optimal solution to be found but did succeed in the development of a heuristic which forms the basis for solution of large scale delivery problems to this day. The solution is synthesized in a number of stages of aggregation in which suboptimizations are carried out on pairs of points or groups. The number of stages of aggregation is a function of the vehicle capacity and the total demand. In the r^{th} stage of aggregation only those points or groups of points are allowed to pair whose combined demand does not exceed $C/2^{N-r}$. It was noted that the method is heuristic; and a twelve-point problem for which the optimal solution was not found was presented.

Clark and Wright (1964) presented a heuristic method based on the work of Dantzig and Ramser with the added provision that vehicles of differing capacities could be considered. The method can be summarized in the following three step procedure:

1. Assume one truck visits exactly one customer then returns to the terminal.
2. If customers i and j are joined by a link then (a) one truck is eliminated and (b) there is a savings in miles travelled.
3. If link ij is feasible it is added. Otherwise, all other possible links are examined until no more can be added.

Using this procedure the authors were able to produce a better feasible solution for the twelve-point problem of Dantzig and Ramser but were

unable to prove optimality.

Balinski and Quandt (1964) offered a formulation for a problem similar to Dantzig and Ramser's with the objective being a minimization of cost. Their formulation takes the form of a generalized set covering problem which requires the enumeration of all feasible single vehicle routes and then selects an optimal set of these routes which meets all demands. Because all feasible routes must be generated, application of the method is a time-consuming process for problems involving a large number of deliveries. However, the authors were able to formulate a simple theory of dominance which reduced the number of routes to be considered. It was suggested that an integer programming cutting-plane algorithm be used to arrive at a solution and computational experience was reported as shown in Table 1.

Gaskell (1967) presented a comparative survey of five methods of solving the problem defined by Dantzig and Ramser. The first of these methods relied on the subjective judgements of the problem solver and was nonquantifiable; the other four were, essentially, variations on the procedure developed by Clarke and Wright. After solving six sample problems with each of the five methods, Gaskell concluded that (1) a computer-oriented technique performs better than a human-oriented technique and (2) none of the variations on Clarke and Wright's procedure were uniformly superior to the original method.

Hausman and Gilmore (1967) culminated several years of research in the publication of a heuristic which solved a somewhat different problem than had been dealt with before. Their definition is as follows: Each of m customers has a minimum required frequency of delivery which

Table 1. Computational Experience as Reported by Balinski and Quandt

Problem	m	n	n'	Pivots	Cuts
1	5	30	26	9	0
2	8	57	24	7	0
3	8	82	68	22	2
4	9	135	102	142	20
5	9	255	203	26	1
6	11	151	145	42	5
7	11	307	305	36	1
8	15	166	142	43	7
9	15	388	270	23	1
10	15	*	*	200+	*

m = number of points to which deliveries must be made

n = number of routes generated

n' = reduced number of feasible routes

* not reported

may be increased to take advantage of economies in routing. Customers are classified into groups, and when any customer in a group requires a delivery the entire group is serviced. The objective is to construct customer groups in such a way as to minimize total annual delivery cost. The formulation resulted in a complex nonlinear programming problem. What was described by the authors as a complicated heuristic, based on the solution of many travelling salesman problems, was developed; but it was not tested to determine how close to optimality it could come.

Hayes (1967) took a different tack in developing a heuristic method in which the route assignments are generated randomly from a weighted probability distribution. The weighting for each demand point is based on its demand for service, its distance from the terminal and from other demand points, and a random element. The author suggested that since the procedure takes very little time it might be repeated for a number of trials and the best solution kept. The optimal solution for the twelve-point problem of Dantzig and Ramser was found in fourteen out of forty trials.

One of the most recent papers on the delivery problem is that of Christofides and Eilon (1969). As did Gaskell, they attempted to compare the performance of several different procedures. The first of these was a branch-and-bound technique based on the travelling salesman algorithm developed by Little et al. (1963). The procedure works as follows:

1. Assume there will be N single vehicle routes in the final solution.
2. Replace the original terminal by N artificial terminals and prohibit travel between them by setting the distances between

them equal to ∞.

3. Solve an associated travelling salesman problem.

4. Repeat for several values of N and take the best result.

Obviously, optimality cannot be guaranteed. The second procedure was that of Clark and Wright. The third procedure, like the first, was based on the travelling salesman problem. An r-optimal tour was defined to be a tour which could not be improved by removing r links and replacing them with r other links. As r increases the number of combinations which must be checked for improvement increases rapidly. However, it was determined that, in general, a 3-optimal tour provides a good approximation of the true optimal. Thus, the procedure assumes a random tour and from this produces a 3-optimal tour. Comparative computation times for the three procedures were reported as shown in Table 2.

J. F. Pierce has succeeded in developing the most efficient optimal algorithm presented to date. Pierce has written on several aspects of scheduling and vehicle delivery as well as on the development of combinatorial programming algorithms for solving set covering problems. His first paper, coauthored with Hatfield (1966), on the use of the travelling salesman problem in solving production scheduling problems led directly to his first comprehensive paper on vehicle delivery (1967). The major concern in this work was with single route problems with a variety of additional constraints. Pierce was primarily interested in techniques that produce feasible solutions early in order to permit premature termination with a feasible solution at hand.

In a later paper (1968) Pierce abandoned his original concepts

Table 2. Computational Experience as Reported by Christofides and Eilon

Problem	Number of Delivery Points	Times (in seconds) Method 1	Method 2	Method 3
1	6	90	6	6
2	13	900	6	6
3	21	-	6	36
4	22	-	6	30
5	29	-	12	48
6	30	-	12	48
7	32	-	12	48
8	50	-	36	120
9	75	-	78	240
10	100	-	150	600

for the development of a combinatorial algorithm for generalized set covering problems. He used a set of delivery problems to test the algorithm. The results are shown in Table 3. Pierce's latest paper (1970) developed certain modifications of the algorithm which improved its efficiency.

Statement of the Problem

The vehicle delivery problem has been shown to possess any of a number of distinguishing characteristics. The particular problem to be considered here is similar to that encountered by Dantzig and Ramser and may be stated as follows: Consider m points each with a demand for deliveries of q_i and a terminal point with no demand. Let C be the capacity of the vehicles and assume that

$$\max q_i < C < \sum_{i=1}^{n} q_i .$$

Further, assume that a symmetric distance matrix $D = [d_{ij}]$, which indicates the distance from any point i to the terminal and to any other point, is known. Find the routing of the vehicles which will satisfy all demands without violating the capacity constraint on the vehicles while minimizing the total cost of deliveries. Two types of costs will be considered--a cost per mile of vehicle travel and a fixed cost incurred for each delivery made by a vehicle.

Objectives of the Research and Method of Attack

Nearly all the authors who have discussed the delivery problem to date have characterized it as relatively simple to formulate but difficult to solve optimally. For this reason a great many heuristics have been

Table 3. Computational Experience as Reported by Pierce

Problem	m	n	Times (in seconds)
1	5	31	.050
2	6	62	.117
3	8	92	.200
4	13	91	6.367
5	11	231	1.383
6	11	561	2.876
7	11	1023	14.383
8	11	1485	19.317
9	12	298	3.500
10	12	538	7.117
11	12	793	4.567
12	15	575	69.483
13	19	1159	2400.000*

m = number of points to which deliveries must be made

n = number of feasible routes examined

* termination without proving optimality

developed but very few algorithms presented are capable of producing optimal solutions. The primary objective of the research reported here was the development of an algorithm which could guarantee an optimal solution to the problem.

The idea of using a two-phase algorithm in which the first phase accomplishes the generation and pricing of feasible routes and the second phase selects an optimal subset of these routes seems ideally suited to the delivery problem. In order to produce such an algorithm one must determine an efficient means for generating feasible routes which keeps the number of routes to be considered minimal and design an efficient method for obtaining an optimal solution to the problem given the set of feasible routes.

Having accomplished the task of developing an algorithm, the second objective of the research lay in examining the computational aspects of the algorithm in an effort to determine areas in which further research might lead to significant modifications.

CHAPTER II

PHASE I

As previously mentioned, the first step in solving the delivery problem is the determination of feasible routes based on delivery quantities and the assignment of costs to each of these routes based on the distance which must be travelled to make the deliveries. This is the function of Phase I of the algorithm.

Generation of Feasible Routes

At the outset, the only information available is the delivery quantities for the points on the schedule, the vehicle capacity, cost information, and the distances between points. Obviously, separate delivery points may be combined to produce multi-delivery routes providing the sum of the requirements of such points does not exceed the capacity of the vehicle. We must then decide how individual deliveries can be aggregated without violating the capacity constraint.

If we represent each feasible route by an m-component binary column vector in which element i takes on a value of one if the route delivers to point i and a value of zero if not, then a simple and efficient scheme for route generation may be developed.

Initially, we create two single-delivery routes, the first delivering to point m and the second to point m-1. Now we wish to determine whether a route delivering to points m and m-1 is feasible. Recalling that route feasibility is solely dependent on the relation

between vehicle load and vehicle capacity, we need only compare the sum of the requirements of points m and m-1 to the vehicle capacity to check the feasibility of such a route. If the route is feasible, it is generated by adding the two original route vectors. If the route is not feasible then we need not consider further routes which deliver to both points m and m-1; and no new route is created. Next, a third single delivery route is created, delivering to point m-2, and we attempt to generate multi-delivery routes by adding this new route vector to each of the vectors previously generated. We continue in this manner until the route which delivers to point the only is created and combined with other possible routes. A proof that this procedure indeed produces all feasible routes appears in Chapter III. Figure 1 illustrates the procedure using the data from Test Problem One as given in Appendix A.

Reduction Theorems

The first major problem encountered in solving the delivery problem concerns the number of feasible routes generated. With $m = 15$, for example, we might generate up to 19,378 feasible routes. While it is true that most practical problems involving fifteen deliveries would allow far less than 19,378 routes, the number of routes may still be a cause for concern.

In the literature, one can find several references to this problem but very few constructive suggestions for circumventing it. Balinski (1965) and Garfinkel (1968) advise the application of four reduction theorems which may eliminate a number of feasible routes from considera-

Create Routes 1 and 2

1	2
0	0
0	0
0	0
0	0
0	1
1	0

Generate Route 3

1	2	3
0	0	0
0	0	0
0	0	0
0	0	0
0	1	1
1	0	1

Create Route 4

1	2	3	4
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	1
0	1	1	0
1	0	1	0

Recursively Apply Procedure

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0	0	0	1	0
0	0	0	1	1	1	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	1
0	1	1	0	0	1	0	0	1	0	0	0	1	0	0	0	0	1	1	0	0	0
1	0	1	0	1	0	0	1	0	0	0	1	0	0	0	0	1	0	1	0	0	0

Figures 1. Procedure for Generating Feasible Routes

tion. Discussion of these theorems requires several simple notational conventions.

One may think of a matrix (in this case, the matrix formed by the route vector) as composed of either n binary column vectors a_j or m binary row vectors r_i. Define the unit vector u_n as the row vector having a one in element n and zeroes elsewhere. Finally, say that $r_k \geq r_t$ if $a_{kj} \geq a_{tj}$ for all j.

Theorem One

If $r_i = \emptyset$ for any i, there is no solution.

This theorem simply requires that each delivery point be included in at least one feasible route vector in order that a solution to the problem exist. Since all single-delivery routes are feasible in the problem under consideration, this theorem will be of no use.

Theorem Two

If $r_k = u_n$ for any k for any n, then a_n must be included in the final schedule.

Here we make the observation that if any point exists which is covered by one and only one route, then that route must appear in the solution to the problem. The comparisons required in the application of this theorem are simple but are not included in the algorithm developed here.

Theorem Three

If $r_k \geq r_t$ then r_k may be deleted as well as any column n for which $a_{kn} = 1$ and $a_{tn} = 0$.

This is to say that if there is any point t to which delivery is made only in conjunction with delivery to some other point k then point

k need not be considered. Further, any route which delivers to point k but not point t need not be considered. Once again, however, we are unable to make use of this theorem since the fact that each point is accorded a single-delivery route precludes satisfaction of the assumptions of the theorem.

Theorem Four

Assume it is not true that $r_k \geq r_t$ or that $r_t \geq r_k$. Let G be the index set associated with the smallest number of vectors u_n such that

$$v_k = r_k + \sum_{n \epsilon G} u_n > r_t.$$

Let

$$w = \bigwedge_{n \epsilon G} a_n$$

where $\wedge$ denotes the "logical and," and let

$$q = v_k - r_t.$$

Then any column p for which $q_p = 1$ and for which there exists a row i such that $a_{ip} = w_i = 1$ may be removed from consideration.

Application of this theorem is best explained through an example. Consider the matrix shown below:

$$\begin{matrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \end{matrix}$$

Were it not for column three, row two would be greater than row one. Thus, through application of Theorem Three, we could omit columns one, two, and five from consideration. If route three is in the schedule, then we must cover point two with route one, two, or five. However, routes two and five both have deliveries in common with route three. Therefore, whether route three is in the schedule or not, routes two and five may be omitted from consideration.

Of the four theorems, only the second and the fourth could prove of use in solving the problem at hand. However, in his discussion of the subject Balinski warns that unless the number of feasible routes is extremely large the time required to apply this theorem cannot be justified.

Pricing Feasible Routes

The generation of feasible routes results in the enumeration of all feasible combinations of deliveries. Before we can attempt to select an optimal delivery schedule, we must assign a cost to each of these combinations.

Recall that two types of costs are to be considered--a cost per mile of vehicle travel and a fixed cost incurred for each delivery made by a vehicle. Since we require that each delivery be made, however, the fixed cost need not be considered, for any feasible schedule will necessarily incur the same fixed cost. Thus, the cost of using a particular route is determined solely by the distance which the route covers.

Consider a particular feasible route, say route j, which makes k deliveries. The distance covered by route j is dependent on the

order in which the k deliveries are made. In fact, we could consider each of the permutations of these k deliveries as separate routes. However, each of these permutations is exactly the same as the others with the exception of the associated cost, and so it is only necessary to consider the least cost permutation. Thus, to assign a cost to route j, we must determine the path that passes through each of the k delivery points once and only once while minimizing the total distance travelled. The problem of finding this path is, of course, the well-known travelling salesman problem. This problem has been treated by a number of different people using a variety of techniques. We must then select the particular technique we will use. In concluding a survey of algorithms for the travelling salesman problem, Bellmore and Nemhauser (1968) made their choice: "If the authors were faced with the problem of finding a solution to a particular travelling salesman problem we would use dynamic programming for problems with 13 cities or less, Shapiro's branch-and-bound algorithm for larger problems. . ."[1] The dynamic programming approach mentioned is that of Bellman (1962).

The test problems which we will consider here relate to the delivery problems of a firm whose customers order material in quantities which comprise at least one-tenth truckload. Thus, no route making more than ten deliveries would ever be considered. This being the case, solving the travelling salesman problem associated with each route by the dynamic programming approach would result in the costs which we desire.

---

[1]Bellmore, M. and G. L. Nemhauser, "The Travelling Salesman Problem: A Survey," Operating Research, Vol. 16, 1968, pp. 538-58.

Of course, for single-delivery routes, there is only one path to consider so the travelling salesman problem need not be solved. This is true for routes making two deliveries also, since we are dealing with a symmetric distance matrix. For three-delivery routes the observations given in Table 4 indicate that it is more efficient to explicitly enumerate all possible paths than to apply the dynamic programming procedure.

Thus, in order to derive route costs, we may directly calculate costs on one and two delivery routes; and we shall enumerate all permutations of deliveries on three-delivery routes and take the least cost permutation. For routes making four deliveries or more, application of Bellman's dynamic programming algorithm will yield the desired result. As we record route costs, we must also record the order in which deliveries are to be made in order to supply an optimal schedule upon completion of the algorithm.

Table 4. Computational Comparison Between Explicit Enumeration (EE) and Dynamic Programming (DP)

n	Additions		Comparisons	
	DP	EE	DP	EE
3	8	6	8	2
4	21	48	21	19
5	84	480	84	234
6	245	2880	245	1433

CHAPTER III

PHASE II

At this point we have generated a set of feasible routes and determined the cost and order of delivery for each route. To furnish an optimal delivery schedule, we must now select a minimum cost subset of these routes which delivers to each point once and only once. This is the function of Phase II of the algorithm.

Mathematical Formulation

The problem of determining an optimal delivery schedule given the routes generated in Phase I may be formulated as a zero-one integer programming problem as shown below.

$$\min \sum_j c_j x_j$$

$$\text{subject to } \sum_j a_{ij} x_j = 1 \quad \text{for } i = 1, 2, \ldots, m \qquad (1)$$

$$x_j = 0 \text{ or } 1.$$

where c_j is the cost of using route j; x_j is a binary decision variable taking on a value of one if route j is used and a value of zero otherwise; and a_{ij} is the i^{th} component of the vector which represents route j.

There are three peculiarities of this formulation which should be noted here: (1) all coefficients in the objective function are positive, (2) all coefficients in the constraint matrix are zero or one, and (3) the right-hand side of the constraint equation is a vector consisting

entirely of ones. These properties are commonly associated with a specific class of zero-one integer programming problems known as generalized set covering problems.

Of course, there are many existing techniques for solving zero-one problems. Most notable are the algorithms of Balas (1965) and Geoffrion (1967). Likewise, the set covering problem has received attention. Gomory's cutting-plane algorithm (1960) has been shown to deal with this formulation most effectively, and the algorithm developed by Pierce has shown promising results. However, it is felt that the special properties of the set covering problem have yet to be fully exploited, and so we will depart from existing techniques in solving the problem here.

Development of the Algorithm

If we drop the integer restrictions on the variables x_j then our formulation reduces to

$$\min \sum_j c_j x_j$$

$$\text{subject to } \sum_j a_{ij} x_j = 1 \quad \text{for } i = 1, 2, \ldots, m \tag{2}$$

$$x_j \geq 0.$$

This is, of course, a linear programming problem which retains all the special properties of the generalized set covering problem mentioned above. Solving this problem by some existing linear programming technique, such as the simplex method, will yield a solution in which the values of the basic variables will fall in the interval $[0,1]$. Denote

the objective function value of this solution by z_0^*. Then z_0^* is a lower bound on the value we could have obtained had we retained the integer restrictions.

If the solution to (2) is all integer, it is the optimal solution to (1), and we need proceed no further. Unfortunately, we cannot guarantee that the solution to the linear programming problem will be all integer. In this case, it would seem logical to examine the effect of setting one of the variables x_j in (2) equal to zero or one. But which variable should we choose?

The variables x_j represents a yes-no decision on the use of route j. Thus, if we could somehow determine a route j which we would like to force into the solution, then the corresponding x_j should be set to one. The desirability of using a particular route is determined by the cost of that route, and so it would seem that we would be working in the right direction by requiring the use of the minimum cost route. Suppose, however, that routes k and p had the same cost and that this cost was lower than that associated with all other routes. Suppose, further, that route k made more deliveries than route p. Then obviously we would prefer to use route k since it completes more of the schedule than route p and at the same cost. Thus, rather than selecting the minimum cost route, it is more reasonable to select the route that minimizes the ratio of cost to number of deliveries made.

The variables which comprise the optimal basis of the linear programming problem just solved represent the set of routes which would be most desirable to use. Confining the search to this set of variables then appears justifiable; and by so doing, a savings in computation time

will be realized.

Having completed this search, the customary way of proceeding would be to add equations to the final tableau of the simplex and derive a solution to the new problem via the dual simplex. However, the opportunity presents itself to take advantage of one of the special properties of our problem. Since we require that each delivery be made exactly once, the act of forcing one route into the solution will necessarily force other routes out of the solution and force the corresponding decision variables out of the problem. More explicitly, any route which makes a delivery that is made by the route which we have chosen to force into the solution nee not be considered. Thus, we may reduce the size of the problem by deleting the variables and vectors in the constraint matrix which correspond to such routes.

One of the niceties of (2) is that the single-delivery routes guarantee the existence of an initial basis for the simplex. After removing a number of variables from the problem, however, no such guarantee can be made. This problem may be overcome by adding artificial variables with very large associated costs.

Another problem becomes apparent at this point. What if the variable which we set to one in the reduced problem took on a value of one in the solution to the larger problem? Obviously, nothing would be gained. In order to prevent this occurrence, we simply delete basic variables with integer values from consideration in determining the variable which we wish to fix.

The procedure described above yields a reduced linear programming problem which will have at least one variable at an integer value in the

optimal solution. If this problem is then solved, we will be one step closer to obtaining an all integer solution to the problem.

There can be no guarantee that the solution to the reduced linear programming problem will be all integer. If it is not, the procedure described will have to be repeated. However, care should be taken before moving blindly ahead. Recall that the value of the objective function in the solution to (2), z_0^*, is a lower bound on the optimal integer solution. Likewise, the value of the objective function in the solution to the reduced problem, which we shall denote by z_1^*, is a lower bound on the optimal integer solution under the restriction that the variable which we have fixed takes on a value of one. It is quite likely that z_1^* will be greater than z_0^*. Should this happen, we choose to fall back rather than to proceed with our particular variable being fixed at one. The logical alternative here is to examine the effect of setting the variable to zero rather than one. This is, indeed, what we do. (Some type of backtracking procedure could be substituted for this process.) The process of setting a variable to zero will not afford as great a reduction of the original problem as setting a variable to one. In fact, only this one variable will be removed from the problem. However, having set this variable to zero and solved the resulting linear program, we will have an indication of the desirability of including this variable in the final solution.

Several questions come to mind in light of the discussion to this point. Suppose that we solve the reduced linear programming problem and find that $z_1^* = z_0^*$. In this case, we would proceed by setting another variable to one and further reducing the problem. Is it not possible

that the new variable we set to one represents a route which has a delivery in common with the route corresponding to the previous fixed variable? The answer is no. The search for a new fixed variable is made over the non-integer basis variables only, and these variables at their optimal values satisfy the constraints on the problem. Obviously, a variable assigned a non-integer value could not have a delivery in common with the route forced into the solution.

Suppose, on the other hand, that we find that $z_1^* > z_0^*$. Here we return to the original problem, remove one variable, and solve the corresponding problem. Continuing from this point, we could possibly encounter a situation in which it would be desirable to remove another variable from the problem. In such a case, would it not be possible to remove all routes from consideration which make a particular delivery and thereby produce a subproblem with no feasible solution? The answer here is yes. There is no way to avoid running into this situation. Still, by making a simple feasibility check before fixing a variable at zero, we can avoid solving a linear program which would prove infeasible. Some means of recording the occurrence of an infeasible combination of fixed variables must be included to prevent exploring this combination of variables again at some later time.

After obtaining the solution to any reduced problem then, we review previous computations to determine the combination of fixed variables which has the minimum lower bound. Moving to this point, if it is necessary to move, we establish a new variable to be set to an integer value, make the corresponding reductions on the problem, and solve the resulting linear programming problem.

Proceeding in this manner will ultimately result in the case of an integer solution at the point with the minimum lower bound. When this occurs, the optimal solution to (1) will have been uncovered.

Summary of the Algorithm

The discussion in the preceding section of this chapter rather loosely develops the basic structure of the Phase II algorithm. A more rigorous statement in the form of a step-by-step procedure follows.

1. Using the routes and costs generated in Phase I, formulate the linear programming problem (2).

2. Set k, $\bar{k} = 0$. Solve the linear programming problem.

3. If the solution to the linear programming problem is all integer, then the optimal solution to (1) has been found; terminate.

4. Record the values z_k^*, the optimal objective function value, and $x_{k,i}^*$, the values of the variables in the optimal basis.

5. Search over the variables in the optimal basis of problem $\bar{k}$ to find the variable $x_{\bar{k},s}^*$ which has a non-integer value and which minimizes the ratio of cost to number of stops made.

6. Make the problem reductions which correspond to fixing the variable $x_{\bar{k},s}^*$. (a) If $x_{\bar{k},s}^*$ has not been previously fixed, then we wish to set $x_{k+1,s} = 1$. Form the reduced constraint matrix by taking the vector A_s and the vectors corresponding to any other variables fixed to one. Add to the matrix all other vectors which do not conflict with these mandatory route vectors. Finally, add those single-delivery route vectors whose delivery points are covered by routes fixed at one and set the associated costs equal to M, where M is a large number.

(b) If $x_{\bar{k},s}^*$ has been previously fixed, then we wish to set

$x_{k+1,s} = 0$. If the vector A_s does not represent a single-delivery route, remove this vector from the constraint matrix; otherwise, simply set $c_s = M$. Determine whether this problem has a feasible solution by calculating $\sum_j a_{ij}$ for each i. If this sum is zero for any i then set $z^*_{k+1} = M$ and go to step 9.

7. Set $k = k+1$. Solve the reduced linear program.

8. Record the values z^*_k and $x^*_{k,i}$.

9. Determine the value of k for which z^*_k is a minimum. Denote this value by $\bar{k}$. If the $x^*_{\bar{k},i}$ are all integer, terminate; if this point has initiated two previous branches, set $z^*_{\bar{k}} = M$ and repeat this step; otherwise, go to step 5.

This procedure is illustrated in Figure 2.

Finiteness and Convergence

The proof of finiteness concerns only the Phase II algorithm since the algorithm of Phase I obviously terminates after a maximum of $3 \times 2^{m-2} + \sum_{i=1}^{m-3} 2^i + 1$ routes have been generated and their costs established. Convergence, however, is dependent upon both phases.

In order to assure that the algorithm terminates in a finite number of iterations--where one iteration is taken to include the steps required to set up and solve one linear programming problem--examine the method by which variables are fixed. Each of the n variables may be fixed at a value of zero or one. The first variable to be fixed will initially be set at a value of one and perhaps later at value zero but cannot be fixed more than twice. The second variable to be fixed may be set four times, once at one and once at zero for each of the two values of the initial fixed variable. The third fixed variable may

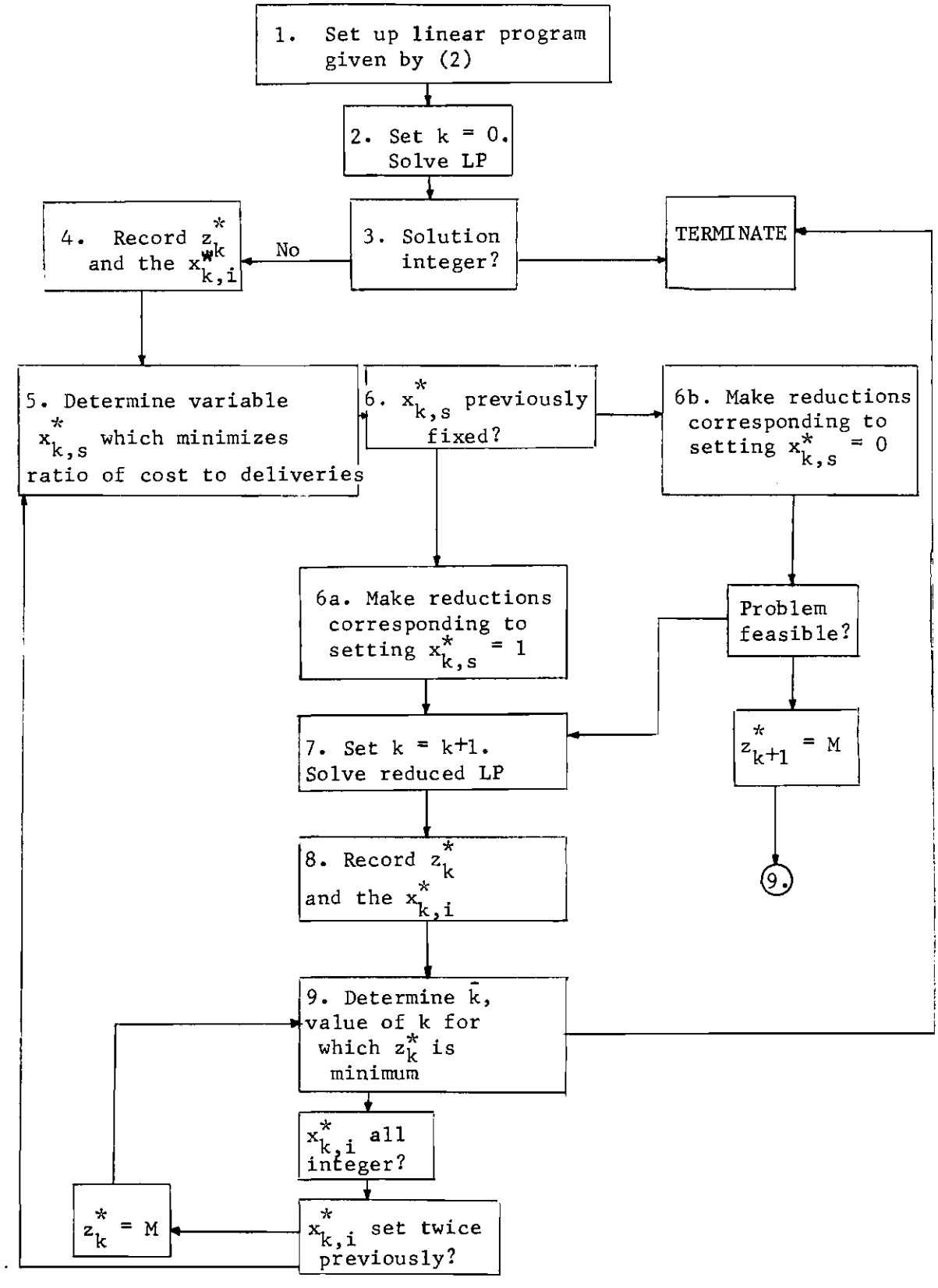


Figure 2. Summary Presentation of Phase II Algorithm

then be set a total of eight times, the fourth a total of sixteen times, and so on. Each time a variable is fixed, one iteration is made. Should every variable be set as many times as possible then, only a finite number of iterations, $\sum_{i=0}^{n} 2^{n-i}$, would be made.

Before showing that the Phase II algorithm will indeed uncover the optimal solution to the formulation (1), it is necessary to insure that the set of routes generated in Phase I includes every possible feasible route.

In order to accomplish this introduce the notation A_{ij} is a route vector with its first nonzero entry in element j and which includes i nonzero entries or deliveries. Suppose now that there exists some route vector A_{ij} which is feasible but which was not generated in Phase I. Obviously, this vector A_{ij} can be expressed as the sum of two other route vectors $A_{1,j}$ and $A_{i-1,k}$ where $A_{i-1,k}$ is a route vector which includes every delivery made by A_{ij} except the j^{th}. The vector $A_{1,j}$ was generated of necessity since it represents a single-delivery route. Thus, because we generated routes by attempting to add the vectors representing previously generated routes, A_{ij} would have been generated had $A_{i-1,k}$ been generated. So we may conclude that the vector $A_{i-1,k}$ was not generated. If $A_{i-1,k}$ is infeasible then A_{ij} would be infeasible so we may further conclude that $A_{i-1,k}$ is feasible. By applying a similar argument to the vector $A_{i-1,k}$ that we used with A_{ij} we may conclude that a third feasible route vector, say $A_{i-2,l}$, was not generated. By recursively applying this argument, we may ultimately conclude that the vector $A_{1,m}$ was never generated. However, the first step of Phase I is the generation of the vector $A_{1,m}$. Therefore, the

vector A_{ij} must be infeasible.

Now that we can be sure that the formulation (1) accurately describes the problem, it is necessary to prove that the Phase II algorithm will find the optimal solution to (1). If the solution to (2) is all-integer, this fact is obvious. The case of concern is then if the solution to (2) is noninteger. If this is the case, we proceed by setting a variable to an integer value and determining a lower bound on the best all-integer solution we may obtain with this new restriction. Continuing in this manner, the algorithm terminates only when we have obtained an integer solution whose objective function value is lower than the lower bounds obtained with all the other partial integer solutions which have been enumerated. Obviously, at this point we will have the optimal integer solution to (2) which is, of course, the optimal solution to (1). We must guarantee, however, that at some time an all-integer solution will be found. Again, this point is obvious. If, for instance, we never move back to set a variable to zero but continue setting variables to one then an all-integer solution must occur after at most m iterations. The fact that we do not proceed in exactly this manner makes no differences for we will still move in the direction indicated above though perhaps in a more roundabout manner.

CHAPTER IV

COMPUTATIONAL ASPECTS OF THE ALGORITHM

To investigate the computational feasibility of the algorithm, computer programs were written in Algol 60 and a number of test problems run on a Burroughs 5500 computer. The results of these tests as well as a discussion of certain extensions of the algorithm, for which no computational experience has been gained, are presented below.

Computational Experience

In Table 4 is shown the solution times for a set of ten test problems some of which were taken from the literature--the others arising from the shipping requirements of an actual firm. The data for these problems appear in Appendix A. Problem 9 was taken from Pierce and Problem 8 is the twelve-point problem of Dantzig and Ramser.

To obtain an indication of the efficiency of the algorithm relative to existing techniques, note that Pierce obtained an optimal solution to Problem 9 in 3.50 seconds and the optimal solution to Problem 8 has been found by Christofides and Eilon in 900 seconds and by Clarke and Wright in six seconds. All of these results were obtained on an IBM 7094 computer.

As seen from the solution times in Table 4, problem-solving time tends to increase both with the number of routes n and the number of constraints m in the problem. The time requirement for any given problem, however, is quite unpredictable. For example the solution

Table 5. Solution Times for Test Problems

Problem number	m	n	Iterations	Solution times (in seconds) Phase I	Phase II	Total
1	6	22	5	2.63	3.13	5.76
2	5	24	7	1.87	8.01	9.88
3	8	83	11	11.82	49.00	60.82
4	8	68	1	14.38	0.25	14.63
5	10	100	3	17.58	17.77	35.35
6	10	155	9	27.32	44.18	71.50
7	10	122	12	22.46	89.38	111.34
8	12	298	6	57.35	59.47	116.82
9	15	139	4	21.67	16.11	37.78

time for Problem 7 is substantially greater than that for Problem 9 even though the deliveries in the smaller problem are a subset of those in the larger.

One important attribute of the algorithm has not been previously mentioned--this being that a feasible integer solution to a problem may be obtained much sooner than the optimal solution. The importance of this aspect can be seen in Problem 7 where the optimal solution was not found until 111.34 seconds had passed but the algorithm could have been terminated any time after 65.45 seconds with a feasible solution.

Returning now to the efficiency of the algorithm, it should be pointed out that no real conclusions can be drawn on the basis of the limited computational experience gained to date. Further, any comparisons between this algorithm and other algorithms are greatly hampered by the fact that the differences in computing machinery used are rather large.

Extensions of the Algorithm

As was stated earlier, the vehicle delivery problem may exhibit any of a number of distinguishing characteristics. The algorithm which we have presented here was developed specifically for the problem as stated in Chapter I; however, by making certain modifications, the algorithm can be extended to handle a variety of different restrictions or assumptions.

Multiproduct Considerations

Consider, first of all, the situation in which more than one product must be delivered. If the products are homogeneous--that is,

they may be loaded on vehicles without discrimination--we need only specify individual delivery quantities as the total of all products and proceed with the algorithm as stated. Nonhomogeneous products, on the other hand, require certain modifications. One possible approach in this case would be to divide delivery point i into k separate delivery point if there are k products to be delivered to point i. After generating all feasible single-product routes, we would then attempt to combine routes which deliver different products according to capacity restrictions and the restrictions placed on shipping the individual products together.

Limited Number of Vehicles

Here no modification to the Phase I algorithm is required. In formulating the problem in Phase II, however, one additional constraint of the form

$$x_1 + x_2 + x_3 + \ldots + x_n \leq V$$

where V is the number of available vehicles, must be added.

Varying Vehicle Capacities

When dealing with vehicles that are nonhomogeneous in the sense that capacities vary, a simple extension of Phase I will allow use of the algorithm. Suppose, for instance, we have vehicles with N_c different capacities. This is handled by repeating the process of route generation N_c times, varying the vehicle capacity each time. Obviously, a large number of routes will be generated, and it would be desirable to reduce this number if possible. Such a reduction can be made by comparing routes formed for the various classes of vehicles. Speci-

fically, if identical routes are made by several classes of vehicles then only that route with the lowest associated cost need be considered.

Distance Constraints

Quite often, especially when a firm must seek the services of a contract shipper, restrictions on the maximum route distance may appear. These restrictions are easily handled by making the appropriate comparisons during the process of route generation.

Optional Deliveries

If certain deliveries are specified as optional, the problem takes on a new dimension. This consideration should not generally pose a great problem, however. By replacing the quality constraints in the formulation of Phase II with inequality constraints most such cases can be dealt with.

Time Constraints

Especially when dealing with perishable products, it is mandatory that we consider the amount of time which will elapse before delivery is made. In such cases, the measure of effectiveness of a particular delivery schedule is expressed in units of time rather than cost. If we desire to specify that a particular delivery be made before a certain time, we simply make the appropriate comparisons while generating feasible routes.

Minimizing Number of Vehicles Used

The restrictions mentioned to this point have dealt exclusively with the nature of deliveries rather than the objective in solving the problem. It may be the case, however, that we are not so concerned with the cost of delivery as with the number of vehicles used. This

consideration may be effectively handled by eliminating the pricing procedure in Phase I and simply assigning a cost of one to each route.

CHAPTER V

CONCLUSIONS AND RECOMMENDATIONS

The conclusions which can be drawn from the research involved in the development of the algorithm lie in two areas: (1) those dealing with the use of the two-phase algorithm for solving delivery problems and (2) those concerning the use of the Phase II algorithm alone for solving general zero-one integer programming problems. Since the research was concentrated in this first area, we will begin there.

Because most other existing techniques are heuristic in nature, it is difficult to make comparisons. However, the two-phase algorithm performed for the most part on a level equal to or better than all existing techniques, with the notable exception of Pierce's algorithm. The criterion used in making this comparison is time required to obtain a final solution. Further, since optimality can be guaranteed, we would tend to appraise this algorithm as superior to many of the others. Additional experience must be gained, however, before any conclusive comparisons can be made. The experience gained to date indicates that this additional experience is justifiable.

Due to the variety of characteristics of the delivery problem, it would definitely be worthwhile to explore the modifications of the algorithm proposed in Chapter IV. Since each type of delivery problem exhibits its own special structure, further research should result in the development of a variety of schemes to take advantage of these

peculiarities.

Unfortunately, no results have been obtained to date which would indicate the effect, computationally, of incorporating any of the modifications mentioned above in the algorithm. However, these modifications require more computations in generating and pricing route vectors but fewer computations in Phase II since fewer feasible routes will be generated. Additional computational experience should bear this out.

The linear programming code used in this work is felt to be somewhat less than perfect. Experiements with other codes should, therefore, serve to enhance the efficiency of the algorithm. Again the promise which such experiments hold should justify their being carried out.

Aside from providing a means for solving the delivery problem, the research has introduced a somewhat different approach to the branch-and-bound technique for integer programming. Results to date indicate this approach to be on a par with many of the existing integer programming techniques. The advantages of pursuing this line of application are many, though it is felt that substantial modifications will be required to move outside the realm of the generalized set covering problem.

In summary, the algorithm presented here appears to hold promise, both as a technique for the delivery problem and as a technique for zero-one integer programming. It is recommended that additional research be carried out in relation to improving the existing algorithm and extending it to the other cases of the delivery problem.

APPENDIX A

TEST PROBLEM DATA

Problem One

Distance Matrix								Demand
	0	1	2	3	4	5	6	
1	78							.36
2	80	54						.44
3	82	9	44					.52
4	89	30	10	39				.40
5	91	42	25	21	18			.32
6	98	48	19	34	9	12		.32

Problem Two

Distance Matrix							Demand
	0	1	2	3	4	5	
1	25						.20
2	15	10					.20
3	3	1	4				.40
4	14	4	2	15			.30
5	5	5	5	5	3		.40

Problem Three

Distance Matrix										Demand
	0	1	2	3	4	5	6	7	8	
1	41									.14
2	38	3								.44
3	80	54	56							.32
4	80	54	56	3						.48
5	97	64	67	19	16					.32
6	92	59	62	13	10	7				.32
7	96	56	59	16	14	11	10			.10
8	78	39	41	54	54	53	57	48		.34

Problem Four

Distance Matrix										Demand
	0	1	2	3	4	5	6	7	8	
1	91									.30
2	98	58								.30
3	96	59	62							.28
4	40	45	5	60						.60
5	73	34	37	46	46					.18
6	82	48	46	44	44	45				.12
7	55	16	19	54	54	55				.56
8	52	16	17	68	68	77	72	69		.34

Problem Five

Distance Matrix												Demand
	0	1	2	3	4	5	6	7	8	9	10	
1	91											.30
2	98	58										.30
3	96	59	62									.28
4	40	45	5	60								.60
5	73	34	37	46	46							.18
6	82	48	46	44	44	45						.12
7	55	16	19	54	54	58	55					.56
8	52	16	17	68	68	77	72	69				.34
9	76	46	49	8	9	19	15	18	44			.52
10	76	44	46	11	11	20	15	15	42	19		.48

Problem Six

Distance Matrix												Demand
	0	1	2	3	4	5	6	7	8	9	10	
1	52											.34
2	76	46										.52
3	76	44	46									.48
4	76	50	53	4								.18
5	72	33	34	53	54							.12
6	98	58	61	53	30	27						.12
7	98	58	61	32	29	26	29					.64
8	93	66	68	14	12	23	18	22				.48
9	89	55	58	10	9	8	5	7	48			.28
10	68	32	33	64	64	67	66	57	22	55		.10

Problem Seven

Distance Matrix												Demand
	0	1	2	3	4	5	6	7	8	9	10	
1	38											.44
2	80	56										.32
3	80	56	3									.48
4	96	67	19	16								.32
5	92	62	13	10	7							.32
6	78	41	54	54	53	57						.34
7	98	62	20	17	12	13	39					.22
8	95	61	15	12	8	8	45	6				.22
9	91	53	25	22	24	19	30	9	15			.30
10	98	61	19	16	10	10	42	3	4	12		.30

Problem Eight

Distance Matrix														Demand
	0	1	2	3	4	5	6	7	8	9	10	11	12	
1	9													.20
2	14	5												.28
3	21	12	7											.25
4	23	22	17	10										.23
5	22	21	16	21	19									.28
6	25	24	23	30	28	9								.23
7	32	31	26	27	25	10	7							.20
8	36	35	30	37	35	16	11	10						.32
9	37	37	36	43	41	22	13	16	6					.30
10	42	41	36	31	29	20	17	10	6	12				.27
11	50	49	44	37	31	28	25	18	14	12	8			.28
12	52	51	46	39	29	30	27	20	16	20	10	10		.18

Problem Nine

Distance Matrix														Demand
	0	1	2	3	4	5	6	7	8	9	10	11	12	
1	38													.44
2	80	56												.32
3	80	56	3											.48
4	97	67	19	16										.32
5	92	62	13	10	7									.32
6	78	41	54	54	53	57								.34
7	98	62	20	17	12	13	39							.22
8	95	62	15	12	8	8	45	6						.22
9	91	53	25	22	24	19	30	9	15					.30
10	98	61	19	16	10	10	42	3	4	12				.30
11	96	62	17	14	8	8	44	5	2	14	2			.28
12	40	5	60	61	71	65	40	66	65	54	65	66		.60

Problem Ten

Distance Matrix

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Demand
1	38																.44
2	80	56															.32
3	80	56	3														.48
4	78	41	54	54													.34
5	91	53	25	22	30												.30
6	98	61	19	16	42	12											.30
7	96	62	17	14	44	14	2										.28
8	40	5	60	61	40	54	65	66									.60
9	55	19	54	54	22	38	48	50	18								.56
10	52	17	68	68	36	57	67	69	14	21							.34
11	76	49	8	9	44	24	20	18	52	45	59						.52
12	76	46	11	11	42	18	20	18	47	39	57	6					.48
13	98	61	32	29	28	7	16	18	60	44	61	34	28				.64
14	93	68	14	12	65	32	24	22	72	61	79	20	26	39			.48
15	89	58	10	9	48	18	9	7	62	51	69	15	12	23	15		.28

APPENDIX B

ALGOL PROGRAM CODE

```
BEGIN
PROCEDURE INDATA;
    BEGIN
        READ(INPUT,/,M);
        READ(INPUT,/,FOR I←1 STEP 1 UNTIL M DO Q[I]);
        READ(INPUT,/,FOR I←0 STEP 1 UNTIL M-1 DO
            FOR J←I+1 STEP 1 UNTIL M DO D[I,J]);
    END OF INDATA;
PROCEDURE MAKEROUTE;
    BEGIN
      LABEL L1,L2;
        A[M,1]←1;
        R[1]←Q[M];
        A[M-1,2]←1;
        R[2]←Q[M-1];
        N←2;
        ROW←M-1;
L1;
        NEW←1;
        FOR I←1 STEP 1 UNTIL N-1 DO
        IF R[I]+R[N]≤1 THEN
    BEGIN
        R[N+NEW]←R[I]+R[N];
        FOR K←1 STEP 1 UNTIL M DO
        IF A[K,I]≥A[K,N] THEN
        A[K,N+NEW]←A[K,I] ELSE
```

```
            A[K,N+NEW]←A[K,N];
            NEW←NEW+1;
        END;
            N←N+NEW;
            ROW←ROW-1;
            IF ROW<1 THEN GO TO L2;
            A[ROW,N]←1;
            R[N]←Q[ROW];
            GO TO L1;
L2;
            N←N-1;
        END OF MAKEROUTE;
PROCEDURE PRICEROUTE;
        BEGIN
            FOR J←1 STEP 1 UNTIL N DO
        BEGIN
            K←0;
            ST[J]←0;
            FOR I←1 STEP 1 UNTIL M DO
            IF A[I,J]=1 THEN
        BEGIN
            ST[J]←ST[J]+1;
            K←K+1;
            G[J,K]←I;
        END;
        END;
            FOR J←1 STEP 1 UNTIL N DO
        BEGIN
            IF ST[J]=1 THEN
            C[J]←2×D[0,G[J,1]];
            IF ST[J]=2 THEN
```

```
      C[J]←D[0,G[J,1]]+D[0,G[J,2]]+D[G[J,1],G[J,2]];
      IF ST[J]=3 THEN
BEGIN
      CP[1]←D[0,G[J,1]]+D[G[J,1],G[J,2]]
           +D[G[J,2],G[J,3]]+D[0,G[J,3]];
      CP[2]←D[0,G[J,1]]+D[G[J,1],G[J,3]]
           +D[G[J,2],G[J,3]]+D[0,G[J,2]];
      CP[3]←D[0,G[J,2]]+D[G[J,1],G[J,2]]
           +D[G[J,1],G[J,3]]+D[0,G[J,3]];
      C[J]←CP[1];
      IF CP[2]<C[J] THEN
BEGIN
      C[J]←CP[2];
      ROW←2;
END;
      IF CP[3]<C[J] THEN
BEGIN
      C[J]←CP[3];
      ROW←3;
END;
      IF ROW=2 THEN
BEGIN
      ROW←G[J,2];
      G[J,2]←G[J,3];
      G[J,3]←ROW;
END;
      IF ROW=3 THEN
BEGIN
      ROW←G[J,1];
      G[J,1]←G[J,2];
      G[J,2]←ROW;
```

```
    END;
    END;
        IF ST[J]≥4 THEN
    BEGIN
        FOR I←1 STEP 1 UNTIL ST[J] Do
    BEGIN
        FOR K←1 STEP 1 UNTIL ST[J] DO
        IF I≠K AND I≠FSTAR[I-1,K] THEN
    BEGIN
        F[I,K]←D[G[J,I],G[J,K]]+FSTAR[I-1,K];
        FSTAR[I,K]←F[I,U];
        FOR ZZ←1 STEP U UNTIL ST[J] DO
        IF F[I,ZZ]<FSTAR[I,K] THEN
        FSTAR[I,K]←F[I,ZZ];
    END;
    END;
        MIN←FSTAR[ST[J],1];
        FOR K←2 STEP 1 UNTIL ST[J] DO
        IF MIN>FSTAR[ST[J],K] THEN MIN←FSTAR[ST[J],K];
        G[J,1]←MIN;
        FOR K←2 STEP 1 UNTIL ST[J] DO
        G[J,K]←FSTAR[K,G[J,K-1]];
     END;
    END OF PRICEROUTE;
PROCEDURE PREPARELP1;
    BEGIN
        FOR J←1 STEP 1 UNTIL N DO
    BEGIN
        NXJ[J]←J;
        VAR[J]←J;
    END;
```

```
          I←1;
          FOR J←1 STEP 1 UNTIL N DO
          IF ST[J]=1 THEN
     BEGIN
          NXI[I]←J;
          I←I+1;
     END;
          FOR I←1 STEP 1 UNTIL M DO
          FOR J←1 STEP 1 UNTIL N DO
     BEGIN
          AA[I,J]←A[I,J];
          CJ[J]←C[J];
     END;
          FOR I←1 STEP 1 UNTIL M DO B[I]←1;
     END OF PREPARELP1;
PROCEDURE SOLVELP;
     BEGIN
       LABEL LP2,LP3,LP5,LP6,LP7,LP8,LMIN;
          ROW←0;
          COL←0;
          OPT←FALSE;
          FOR I←1 STEP 1 UNTIL M DO
     BEGIN
          FOR J←1 STEP 1 UNTIL N DO
     BEGIN
          IF NXI[I]=NXJ[J] THEN
          CI[I]←CJ[J];
     END;
     END;
          ITER←0;
LP2:
```

```
          FOR J←1 STEP 1 UNTIL N DO
     BEGIN
          Z[J]←0;
          FOR I←1 STEP 1 UNTIL M DO
          Z[J]←(Z[J]+CI[I]×AA[I,J]);
          ZC[J]←Z[J]-CJ[J];
     END;
          OBJ←0;
          FOR I←1 STEP 1 UNTIL M DO
          OBJ←OBJ+(CI[I]×B[I]);
LP3:
          IF OPT THEN GO TO LMIN;
          ITER←ITER+1;
          ZCM←ZC[1];
          JM←1;
          FOR J←2 STEP 1 UNTIL N DO
     BEGIN
          IF ZC[J]≤ZCM THEN GO TO LP6;
LP5:
          ZCM←ZC[J];
          JM←J;
LP6:
     END;
          IF ZCM≤0 THEN GO TO LP7 ELSE GO TO LP8;
LP7:
          OPT←TRUE;
          GO TO LP3;
LP8:
          XM←(1.0×(10*40));
          IM←0;
          FOR I←1 STEP 1 UNTIL M DO
```

```
    BEGIN
         IF AA[I,JM]>0 THEN
    BEGIN
         XX←B[I]/AA[I,JM];
         IF XX<XM THEN
    BEGIN
         XM←XX;
         IM←I;
    END;
    END;
    END;
         XX←AA[IM,JM];
         B[IM]←B[IM]/XX;
         FOR J←1 STEP 1 UNTIL N DO
         AA[IM,J]←AA[IM,J]/XX;
         FOR I←1 STEP 1 UNTIL M DO
    BEGIN
         IF I≠IM THEN
    BEGIN
         XX←AA[I,JM];
         B[I]←B[I]-(XX×B[IM]);
         FOR J←1 STEP 1 UNTIL N DO
         AA[I,J]←(AA[I,J]-(XX×AA[IM,J]));
    END;
    END;
         ROW←NXI[IM];
         COL←NXJ[JM];
         CI[IM]←CJ[JM];
         NXI[IM]←NXJ[JM];
         GO TO LP2;
LMIN:
```

```
    END OF SOLVELP;
PROCEDURE SOLUTIONSAVE;
    BEGIN
        ZSTAR[LPIT]←OBJ;
        FOR I←1 STEP 1 UNTIL M DO
    BEGIN
        XSTAR[LPIT,I]←B[I];
        WX[LPIT,I]←VAR[NXI[I]];
    END;
    END OF SOLUTIONSAVE;
PROCEDURE ACTIVENODE;
    BEGIN
        OBJ←ZSTAR[LPIT];
        NODE←LPIT;
        IF LPIT≥1 THEN
    BEGIN
        FOR I←0 STEP 1 UNTIL LPIT-1 DO
        IF ZSTAR[I]<OBJ AND COUNT[I]≤1 THEN
    BEGIN
        OBJ←ZSTAR[I];
        NODE←I;
    END;
        FOR J←1 STEP 1 UNTIL NN DO
        IF SETV[NODE,J]=1 THEN
        SETV[LPIT+1,J]←1 ELSE
        IF SETV[NODE,J]= -1 THEN
        SETV[LPIT+1,J]← -1;
    END;
    END OF ACTIVENODE;
 PROCEDURE VARYSET;
    BEGIN
```

```
LABEL LDET;
          FOR I←1 STEP 1 UNTIL M DO
          IF XSTAR[NODE,I]≥.00001 AND
          XSTAR[NODE,I]≤.99999 THEN
     BEGIN
          MIN←C[WX[NODE,I]]/ST[WX[NODE,I]];
          K←I;
          SET←WX[NODF,I];
          GO TO LDET;
     END;
LDET;
          FOR I←K+1 STEP 1 UNTIL M DO
          IF MIN>C[WX[NODE,I]]/ST[WX[NODE,I]]
          AND XSTAR[NODE,I]≥.00001 AND
          AND XSTAR[NODE,I]≤.99999 THEN
     BEGIN
          SET←WX[NODE,I];
          MIN←C[WX[NODE,I]]/ST[WX[NODE,I]];
     END;
     END OF VARYSET;
PROCEDURE ONESET;
     BEGIN
          COUNT[NODE]←COUNT[NODE]+1;
          LPIT←LPIT+1;
          IF COUNT[NODE]>1 THEN SETV[LPIT,SET]← =1 ELSE
          SETV[LPIT,SET]←1;
          FOR I←1 STEP 1 UNTIL M DO SUM[I]←0;
          K←1;
          FOR J←1 STEP 1 UNTIL NN DO
          IF SETV[LPIT,J]=1 THEN
     BEGIN
```

```
      FOR I←1 STEP 1 UNTIL M DO
BEGIN
      AA[I,K]←A[I,J];
      SUM[I]←SUM[I]+A[I,J];
END;
      VAR[K]←J;
      CJ[K]←C[J];
      K←K+1;
END;
      N←K;
      FOR J←1 STEP 1 UNTIL NN DO
      IF SETV[LPIT,J]=0 THEN
BEGIN
      ROW←0;
      FOR I←1 STEP 1 UNTIL M DO
BEGIN
      SUM1[I]←SUM[I]+A[I,J];
      IF SUM1[I]>1 THEN ROW←1;
END;
      IF ROW=0 THEN
BEGIN
      FOR I←1 STEP 1 UNTIL M DO
      AA[I,N]←A[I,J];
      VAR[N]←J;
      CJ[N]←C[J];
      N←N+1;
END;
END;
      FOR J←1 STEP 1 UNTIL NN DO
      IF SETV[LPIT,J]=1 AND ST[J]≠1 THEN
BEGIN
```

```
          FOR I←1 STEP 1 UNTIL M DO
          IF A[I,J]=1 THEN
     BEGIN
          FOR K←1 STEP 1 UNTIL M DO
          IF K=I THEN AA[K,N]←1 ELSE AA[K,N]←0;
          VAR[N]←0;
          CJ[N]←10000;
          N←N+1;
     END;
     END;
          N←N-1;
          I←1;
          FOR J←1 STEP 1 UNTIL N DO
          NXJ[J]←J;
          IF VAR[J]=0 OR ST[VAR[J]]=1 THEN
     BEGIN
          NXI[I]←J;
          I←I+1;
     END;
          FOR I←1 STEP 1 UNTIL M DO B[I]←1;
     END OF ONESET;
LABEL CONT2,CONT1,LOPT;
          INDATA;
          NN←N;
          C[0]←100000;
          ST[0]←1;
CONT2:
          SOLVELP;
          FOR I←1 STEP 1 UNTIL M DO
     BEGIN
          INT[I]←0;
```

```
          IF B[I]<.00001 OR B[I]>.99999 THEN INT[I]←1;
     END;
          FOR I←1 STEP 1 UNTIL M DO
          IF INT[I]=0 THEN GO TO CONT1;
          IF LPIT=0 THEN GO TO LOPT;
          INEGR[LPIT]←1;
CONT1;
          SOLUTIONSAVE;
          ACTIVENODE;
          IF INEGR[NODE]=1 THEN GO TO LOPT;
          VARYSET;
          ONESET;
          GO TO CONT2;
LOPT;
      END.
```

BIBLIOGRAPHY

1. E. Balas. "An Additive Algorithm for Solving Linear Programs with Zero-one Variables." Operations Research. Vol. 13, No. 4. 1965.

2. M. L. Balinski. "Integer Programming: Methods, Uses, Computation." Management Science. Vol. 12, No. 3. 1965.

3. M. L. Balinski and R. E. Quandt. "On an Integer Program for a Delivery Problem." Operations Research. Vol. 12, No. 2. 1964.

4. R. Bellman. "Dynamic Programming Treatment of the Travelling Salesman Problem." Journal of ACM. Vol. 9. 1962.

5. M. Bellmore and G. L. Nemhauser. "The Travelling Salesman Problem: A Survey." Operations Research. Vol. 16. 1968.

6. N. Christofides and S. Eilon. "An Algorithm for the Vehicle-Dispatching Problem." Operational Research Quarterly. Vol. 20. 1969.

7. G. Clarke and J. W. Wright. "Scheduling Vehicles from a Central Depot to a Number of Delivery Points." Operations Research. Vol. 12, No. 4. 1964.

8. G. B. Dantzig and J. H. Ramser. "The Truck Dispatching Problem." Management Science. Vol. 6, No. 1. 1959.

9. R. S. Garfinkel. Optimal Political Districting. College of Business Administration, University of Rochester. Rochester, New York. 1968.

10. W. W. Garvin, H. W. Crandall, J. B. John, and R. A. Spellman. "Application of Linear Programming in the Oil Industry." Management Science. Vol. 3. 1957.

11. T. J. Gaskell. "Bases for Vehicle Scheduling." Operational Research Quarterly. Vol. 18, No. 3. 1967.

12. A. M. Geoffrion. "Integer Programming by Implicit Enumeration and Balas' Method." SIAM Review. Vol. 9. 1967.

13. R. E. Gomory. "Outline of an Algorithm for Integer Solutions to Linear Programs." Bulletin of the American Mathematical Society. Vol. 64. 1958.

14. W. H. Hausman and P. Gilmore. "A Multi-Period Truck Delivery Problem." Transportation Research. Vol. 1, No. 4. 1967.

15. R. L. Hayes. The Delivery Problem. Carnegie Institute of Technology, Graduate School of Industrial Administration. Report MSR 106. 1967.

16. J. D. C. Little, K. G. Murty, D. W. Sweeney, and C. Karel. "An Algorithm for the Travelling Salesman Problem." Operations Research. Vol. 11. 1963.

17. J. F. Pierce. "Direct Algorithms for Truck Dispatching Problems: Part I." Transportation Research. Vol. 3, No. 1. 1969.

18. J. F. Pierce. "On the Truck Dispatching Problem: Part I." IBM Cambridge Scientific Center Report 320-3218. Cambridge, Massachusetts. 1967.

19. J. F. Pierce and M. Hatfield. "Production Sequencing by Combinatorial Programming." Chapter 17 of J. F. Pierce, Operations Research and the Design of Management Information Systems. Technical Association of the Pulp and Paper Industry, New York. 1967.