CR-102713

SEARCH FOR GOOD ALGORITHMS FOR PRACTICAL SOLUTIONS TO DISCRETE OPTIMIZATION PROBLEMS

Supported by NAS8-11189

UNIVERSITY OF TENNESSEE COMPUTING CENTER

THE UNIVERSITY OF TENNESSEE
Knoxville, Tennessee

N70-30118

SEARCH FOR GOOD ALGORITHMS FOR PRACTICAL SOLUTIONS TO DISCRETE OPTIMIZATION PROBLEMS

University of Tennessee
Knoxville, Tennessee

FINAL REPORT

Search for Good Algorithms for Practical Solutions to Discrete Optimization Problems

Supported by NAS8-11189

Prepared by Gordon R. Sherman

University of Tennessee Computing Center

Knoxville, Tennessee, March, 1970

TABLE OF CONTENTS

INTRODUCTION

Problems concerning the optimization of mathematical functions have always been of interest to mathematicians and their solutions are of great practical value. Science and engineering have especially benefitted over the past several hundreds of years from techniques developed by mathematicians(in calculus and related areas). Although the need and practical application for optimization has been apparent, mathematics has furnished comparatively little help, however, in solving problems defined over discrete spaces. Usually these problems have been combinatorial in nature and extremely unyielding to theoretical analysis. At least this is true in so far as the record of mathematical analysis in providing theorems or characterizations which are useful for practical application is concerned.

Interest in discrete optimization took a notable jump with the onset of such fields as Operations Research and Management Science. The success of work done in these areas is particularly dependent upon finding solutions to problems in discrete optimization in the 1940's and early 1950's which was independent of the availability of high speed digital computers. Then, with widespread availability of powerful computers becoming a reality in the 1950's, results in solving discrete optimization problems took on even faster growth. It was not long, however, until the computer essentially reached a limit in aiding researchers in this field. Algorithms were still largely being developed by people who were thinking computationally along traditional lines. The computers were used as though they truly were "large, fast, desk calculators". Obviously, when this line of thinking prevailed, computers essentially reached their limit in value when the ratio of computation necessary for solving a problem in small dimensions over a problem with larger dimensions reached the approximate value of the ratio of computer power to desk calculator power. That is, for example, one could better solve linear programming problems with a computer over desk calculators to about the same degree that the computer was faster than the calculator.

The computer had much more to offer, however, It offered a means to solving problems by methods which not only weren't practical when worked by hand, but weren't even thought of. The Monte Carlo method is an early example of this. The increasing use of simulation by digital computer is another. In this work we have investigated another approach which I choose to call the method of stochastic algorithms. It relies upon probability to circumvent much of the enormous complexity which prevails in combinatories. It provides an underlying theory. It has been shown to be quite powerful in some important situations. It provides practical approach to many difficult problems, and its usefulness will grow as experience is gained in formatting many problems into its rather simple structure. This work is primarily aimed at acquiring some of this necessary experience. Our work took on several other side projects which naturally arose during the course of the study. Work was done on the non-discrete quadratic programming problem and also on the problem of finding all shortest paths through a given network.

This project was worked on by several people at the University of Tennessee Computing Center. Among the students who received support from this project (with their major and degree received or being worked on) are: Alfred K. Hume, B. S. in Engineering Physics; Sherrie L. Ippolito, B. S. in Education; Walter L. Leatherwood, M. S. in Management Science; Suzanne M. Miller, M. S. in Computer Science; John M. Rooker, M. S. in Mathematics; Douglas W. Toppins, M. S. in Management Science.

The Stochastic Algorithmic Approach to Discrete Optimizing Problems.

A. Discrete Mathematical Programming

The mathematical programming problem can be stated as

Determine x_o from a space $\mathcal{X}$ such that

$f(x_o)$ is maximized (or minimized) subject to constraints that x_o belong to some well defined subset, C, of $\mathcal{X}$.
f is called an objective function.

Example 1. Linear Programming

maximize $c_1x_1 + c_2x_2 + \ldots + c_nx_n$

subject to the conditions

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}k_n \leq b_1$$

$$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \leq b_2$$

$$\vdots$$

$$a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \leq b_m$$

and $x_i \geq 0 \qquad i = 1,\ldots,n$

In this case x_o equals the vector $(x_1, x_2 \ldots x_n)$, $\mathcal{X}$ is euclidean n - space and C is a convex polyhedron lying in the positive orthant. f is simply a linear function of the components of x_o.

Example 2. Discrete Linear Programming

In discrete linear programming the problem is usually considered to be the same as the linear programming problem except that the C set is restricted (or intersected) with the non-negative integers.

Example 3. Quadratic Programming

The quadratic mathematical programming problem is maximize

$$\sum_{i=1}^{n} p_i x_i + \sum_{i=1}^{n} \sum_{j=1}^{n} x_i q_{ij} x_j$$

subject

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad i = 1,2,\ldots,m$$

and

$$x_i \leq 0 \qquad i = 1,\ldots,n$$

That is, C is identical to that of Example 1 but the objective function is a quadratic instead of a linear function of $x_1, x_2, \ldots, x_n$. In some important cases, the quadratic function is restricted to the positive definite class.

Example 4. Partitioning problems.

Given a sequence 1,2,...,n (call it $\overline{n}$) and a matrix

$$A = \left\{a_{ij}\right\} i, j=1,\ldots,n = \begin{pmatrix} a_{11} & a_{12} \cdots a_{1n} \\ & \cdot \\ & \cdot \\ & \cdot \\ & a_{n1} \cdots a_{nn} \end{pmatrix}$$

find a k set partition $\Pi = (P_1,\ldots,P_{12})$ of $\bar{n}$ such that

$$f(\pi) = \sum_{i,j\varepsilon P_1} \sum_{i,j\varepsilon P_2} \cdots \sum_{i,j\varepsilon P_k} a_{ij}$$

is minimized.

In this case, f is a set function over all possible k set partitions of $\bar{n}$ and $\mathcal{H}$ is the set of all possible k set partitions of $\bar{n}$. C in the example here is non-existent, but in some cases it could consist of a restriction on the cardinalities of the P_j, $j=1,\ldots,k$.

Example 5. Sequencing Problems

A familiar sequencing problem is;

Given $\bar{n}$, and a distance matrix

$\{a_{ij}\}$ $i,j=1,\ldots,n$, find a permutation p $(P_1,P_2\ldots p_n)$ of the points in $\bar{n}$ such that $f(p) = \sum_{i=1}^{n-1} ap_{i,}p_{ih} + ap_n,p_1$

is minimum.

In this case $\mathcal{H}$ would correspond to all permutations of the first n positive integers and f is a function describing the sum of the distances from point to adjacent point defined by the permutation.

Other sequencing problems are those arising when shortest paths through networks are being sought and will be covered later.

B. Heuristic Algorithms

For many problems in discrete optimization, algorithms have been found which lead to global optimal solutions. Examples of these situations are, the linear programming problem and the (non-discrete) quadratic programming problem. These types of algorithms which lead to exact solutions to problems are deterministic but their practicality varies from one case to another. The linear programming algorithms are very useful. In some other cases, the problems for which solutions are derived, and their corresponding algorithms are of theoretical interest only - that is, their practical usefulness is limited. For this reason many approaches to some of the more complex problem situations have led researchers to develop heuristic or near optimal solutions. That is, algorithms which are facile and relatively simple in construct and which have strong intuitive appeal and which show satisfactory and encouraging results when applied to small problems are sought. Some of the criticism of these approaches are: 1) there is no assurance of finding global optima, 2) there is usually no assurance that the heuristic algorithm will be "near" the global optima nor even a method of measuring how close to global optimality any particular solution is, 3) they lack mathematical concinnity, beauty and rigor, 4) their usefulness can be determined only by comparison with other algorithms applied to identical problems. Out approach in this study has been to explore the advantages of near optimal solution methods and to develop a particular class of them in the stochastic algorithmic method. Also, in view of disadvantage 2) stated above, a method of formally evaluating these differing algorithms is developed and experimental comparisons are made.

C. Stochastic Algorithms

Let $\mathcal{H}$ be a discrete space of points and x be a typical member of $\mathcal{H}$. Assume that f is defined for all $x \varepsilon \mathcal{H}$. Then, for $\mathcal{H}$ and f, a stochastic algorithm consists of a particular neighborhood structure an $\mathcal{H}$ and a probability distribution over $\mathcal{H}$, denoted as P(x). The neighborhood structure which we will call N, has the following properties:

1. For each $x \varepsilon \mathcal{H}$, there is a corresponding a neighborhood n(x) consisting of points in $\mathcal{H}$

2. n(x) contains x

Given $\mathcal{H}$, f and N, the stochastic algorithm is completely defined by a successor structure s(x) which has the following properties:

1. $s(x) \varepsilon n(x)$

2. $f(s(x)) \geq f(y)$ for all $y \varepsilon n\ (x)$

3. $f(s(x)) = f(x) \Rightarrow x = s(x)$

4. $s(x) = x \Rightarrow f(x) \geq f(y)$ for all $y \varepsilon n(x)$

If s(x) = x then x is called a locally optimal point. Obviously, the collection of all locally optimal points contains the global optimal point.

A stochastic algorithm proceeds by selecting x from $\mathcal{H}$ according to probability distribution P. The successor structure is then applied (or computed) until a local optimal point is found. The process is repeated a number of times until a satisfactory probability statement can be made about the likelihood of the global optimal point having been found and/or the relationship between the best observed local optimal point, a bound on the global optimal point, and the cost of further computing. This process further detailed in [1].

Much of the success of the stochastic algorithmic approach depends upon the selection of the neighborhood structure to be used for solving each different problem. For this reason, research into the subject and experimental comparisons with familiar problems are important. The following describes some investigations into this subject.

Stochastic Algorithms Applied to a Partitioning Problem

The partitioning problem as defined above has been worked on by several authors including J. A. Joseph. [2] In this part of the study, two stochastic algorithms were developed and programmed to apply to partitioning problems determined by varying the parameters n and k as defined in Example 3 above. Joseph's algorithm was also programmed and applied to identical problems.

The two stochastic algorithms developed for the partitioning problem are called GRS(I) and GRS(II). GRS(I) proceeds as follows for a given n, k and a_{ij} $i,j = 1,2,\ldots,n$

1) Select a random k set partition π of $\bar{n}$

2) Compute $f(\pi)$

3) Set $i = 1$

4) Alter π to πi $i = 1, 2, \ldots, k$ by moving point i into each of the k sets of π.

5) Compute $f(\pi i)$ and determine such that $f(\pi j) = f(\pi i)$ $i = 1,2,\ldots,k$

6) π' is the successor of π

7) If $f(\pi') = f(\pi)$, π is a locally optimal k set partitions with respect to GRSI

8) Go to 1) unless some pre-chosen stopping criterion is satisfied.

GRSII is identical to GRS(I) except that π' is selected as the first partition found which is better (i.e. has a smaller f value) than π. That is, GRSI is a steepest ascent algorithm, GRSII is a positive ascent algorithm.

Experiments were run for randomly determined matrices for n values equal to 10, 15, 20, 25, 30, 35, 40, 45, 50, 75, 100. It turned out that in every single experiment, the best value found by GRS(I) was at least as good and in about 98% of the cases better than that found with Joseph's algorithm. All computer programming was done by the same programming in FORTRAN and run on the same machine. Some sample tabulations are given below:

	N = 35	Functional Values
K=2	Joseph	GRS(I)
	14280	13973
	14551	13578
	13828	13180
	14013	13255
	13871	13226
	14174	13400
*Average Computer Time	68.0	101.18
K = 10	1177	728
	1036	705
	887	692
	1001	771
	1074	713
	934	676
*Average Computer Time	183.8	118.9
K = 15	370	180
	323	150
	334	193
	456	197
	342	232
	357	198
*Average Computer Time	296.0	111.2

*Average computer time is in 60ths of a second on an IBM 7040.

These results are quite typical of the entire experiment. To briefly summarize, we can say:

1) GRSI consistently outperformed the other algorithms in finding better values for the objective function.

2) The time performance was consistent over various values of n and k. The Joseph algorithm performed best for small values of k (k≤5) but GRSI consistently performed better time wise for k larger than 5.

An example of timing data follows:

	GRSI	Joseph
	N=35	
K = 2	101.18	68.0
5	114.85	113.2
10	118.9	183.8
15	111.2	296.0
	N=50	
K = 2	226.2	134.2
5	261.5	224.2
10	262.4	371.2
15	249.6	508.7

Experiments with the optimal sequencing (or traveling salesman) problem.

The results with the optimal partitioning problem are encouraging and suggest that the stochastic algorithmic approach might be especially useful in attacking combinational type problems. Since combinational problems are notoriously famous for being immune to mathematical analysis but at the same time, very critical to the solution of many problems of the real world, the value of our approach here can have a major impact on the advancement of technology. In order to further study this aspect of discrete optimizing, extensive application and experiments with the optimal sequencing problem was carried out. This problem is defined in Example 5 above.

The problems on which we have worked include: Ten 9-city problems given [3], a 15-city problem, a 20-city problem [4], a 25-city problem [5], a 33-city problem [>], a 42-city problem [6], a 48-city problem [5], and a 57-city problem [7].

A family of algorithms were developed and applied to the problems mentioned above. These algorithms which we have labeled Algo I, Algo II, Algo III, Algo IV and AlgoIV(r). They are described below. They were designed for computational convenience. Algo I, Algo II and Algo III were used in this study only to a limited extent; the Algo IV(r) series was used on all of the problems mentioned above for several values of r. Most of our results are for the Algo IV(r) experiments. In each case an estimate was made of the value of r which would have given the best performance. This, of course, involves hindsight and cannot always be expected to carry over to different traveling salesman problems. However, the evidence presented here indicates that in practical situations, our methods are likely to be valuable.

The Algorithms.

Algo I.

Algo I proceeds as follows: A random permutation p of $\{1,2,\ldots,n\}$ is selected and f(p), the length of tour p, is computed. The permutation p' is derived from p by inverting the first and second elements of p. The f(p') is compared with f(p); if f(p') is less than f(p), p' replaces p and f(p') replaces f(p). Then the second and third elements of the resulting permutation are inverted to form a new permutation p" and again comparison is made. This process is continued until n consecutive interchanges have been checked without a change in the permutation, i.e., without reducing the length of the tour. At this point a locally minimal tour has been found. The tour $\hat{p}$ and its length $f(\hat{p})$ are recorded, another random starting permutation is chosen, and the process repeated.

Algo II.

Algo II is similar to Algo I except that instead of comparing two permutations (i.e., by considering the two permutations of a given pair of adjacent points), six permutations are compared. The six permutations are those obtained from the six permutations of $i, i+1, i+2$ (mod n) for $i = p, 2, \ldots, n$. A locally minimal permutation is one such that no better permutation (i.e., one with a shorter tour length) can be found by permuting any three adjacent points in that permutation.

Algo III.

Let $p^1 = (i_1, i_2, \ldots, i_n)$ be a (randomly selected) permutation of of the integers $p, 2, \ldots n$. Now form a new permutation, say p^2, by interchanging the values of i_1 and i_2 to form p^2(i.e., i_1 in p^2 is the i_2 in p^1 and the i_2 in p^2 is i_1 in p^1). Then compute the tour length $f(p^2)$ and records it value. Next interchange i_2 of p^2 and i_3 of p^2 to form a new permutation p^3. Compute $f(p^3)$ and record. Then interchange i_3 of p^3 and i_4 of p^3 to form p^4. Compute $f(p^4)$ and record, etc. This process continues until $f(p^1), f(p^2), \ldots, f(p^{n-1}), f(p^n) = f\ (p)$ have all been computed. A permutation p_α such that $f(p_\alpha) \leq f(p^i)$, $i = 1, 2, \ldots, n$, is determined and is used as a starting point for another set of comparisons. The next set of comparisons is carried out by starting with the second point in the permutation p_α. This can easily be done by circling the elements of p_α one position to the left and then operating on the resulting permutation in the same manner as described above. The process is continued until n sets of (n-1) permutations have been examined without decreasing f. At this point a locally minimal permutation has been found. Another random permutation is then chosen and the algorithm preceeds to find another local minimal permutation.

Algo IV(r).

Algo IV(1) is just Algo III. Algo IV(2) is an extension of Algo III. When a locally minimal permutation, p, has been found by means of Algo III, two adjacent points in p are moved as a pair being placed into that position and orientation which minimizes the length of the tour. This procedure of moving two adjacent points together continues until no pair of points can be moved so as to decrease the tour length. Algo IV(3) is similar process for triples to adjacent points after Algo IV(2) can find no further improvements, while Algo IV(r) continues until no advantageous move can be made by applying Algo III, Algo IV(2), Algo IV(3),..., Algo IV(r-1).

The computer program of the Algo IV(r) algorithum is described by the following. An initial (random) permutation of the first n positive integers is read in each integer associated with a point (or mode or city).

$(C_1, C_2, \ldots C_n)$ represents an initial "tour" starting at C_1, the city in position 1, proceeding to C_2, the city in position 2, etc.

A parameter s $(1 \leq s \leq r)$ refers to the first s cities in the tour. The algorithm, in effect, removes the first s cities and "re-inserts" them sucessively into positions between C_{+1} through C_n. The total tour distance is calculated at each position. If the tour distance is decreased at one or more positions, as compared with the initial tour distance, the s cities are permanently inserted at that position at which the tour distance decreased the most.

The permutation is then shifted so that the city formerly in position C_{s+1} moves to position C_1 and the process is repeated. The s cities are checked in both orientations. For example, if s = 2, and the position being checked is k, the "backward" orientation is checked first:

$---C_k - C_2 - C_1 - C_{k+1}$

and then the forward orientation:

$---C_k - C_1 - C_2 - C_{k+1}$ ---

If no improvement over the initial tour is effected, the permutation is shifted to the left 1 place, so that the initial C_2 becomes C_1.

Example: If 2 - 1 - 3 - 4 represents an initial tour of 4 cities and no improvement (s=2) is found in the following sequence:

3 - 1 - 2 - 4

3 - 2 - 1 - 4

3 - 4 - 1 - 2

(3 - 4 - 2 - 1) represents the same tour as the initial one.

The permutation is shifted to 1 - 3 - 4 - 2 and the process repeated:

4 - 3 - 1 - 2

4 - 1 - 3 - 2

4 - 2 - 3 - 1

In the program,s is initially set equal to 1 is what we call PHASE I analysis. This phase terminates when no further improvement is made by inserting one city at the various positions; i.e., when, after the initial city is checked, no further improvement is made for n - 1 consecutive applications.

s is then increased by 1, and the process repeated until no further improvement is made by inserting two cities, (in either orientation), similarly for s = 3, etc. After s reaches r or specified maximum value, it is set equal to 1 again.

After the initial setting of s = 1, the process terminates when s - 1 successive applications yield no further improvement. In other words, if some improvement is made at s = 4, say, and the maximum specified value of r = 10, then the algorithm will terminate if no further improvement is found by setting s = 5, 6, 7, 8, 9, 10, 1, 2, 3, at this point, we say that a "local optimum" has been found, and the tour distance and tour vector are recorded.

The above method constitutes a PHASE I analysis. The PHASE II analysis is similar, except that s starts at the specified maximum value r and is incremented by -1 at each stage until it reaches 0, at which time it is reset to the maximum value. Detailed material on these computer programs are available from the author.

Some of the results of these experiments are contained in reference [1]. Some of the more important results of this experiment are the following:

1) The stochastic algorithmic approach has again proven to be very powerful in comparison with other known algorithms. In each case, answers as good or better than any other known answers were found.

2) Important insight into the relationship of the definition of the neighborhoods to the performance of the algorithms were established. In particular, the algorithm Algo I, and Algo II and the series Algo IV(r) revealed that performance is clearly related to the neighborhood size. To see this more clearly one might consider the two extreme cases; a) $n(x) = x$ (i.e. each neighborhood consists of itself) and b) $n(x) = \mathcal{X}$ (i.e. each neighborhood consists of the entire space. Case a) leads to simple random search and is clearly inefficient. Case b) is merely our exhaustive search of the entire space, the impracticality of which leads to the discrete optimization problem in the first place.

3) An especially valuable discovery came from the work with the Algo IV(r) series. These algorithms differ from generalizations of Algo I and Algo II in that the neighborhoods differ not only by size but by shape. The neighborhoods in Algo I and Algo II are compact in the sense that each pair of points are relatively close to each other. For example, if we define the distance between two permutations of $\overline{n}$,

$$p = (1,p_2,p_3\cdots p_n)$$

$$q = (1,q_2,q_3\cdots q_n)$$

as $$d(p,q) = \sum_{i,j} \{|i - j| : p_i = q_j\}$$

it can be shown that the average distance between points in the neighborhoods of Algo IV(r) is much greater than the average distance between pairs of points in the neighborhoods of generalizations of Algo I and Algo II. This is true when the cardinalities of the neighborhoods are approximately the same.

Intuitively one can conclude that the superior performance of the Algo IV(r) series is due largely to the shapes of the neighborhoods. The difference in performance within the Algo IV(r) series is also due largely to differences in neighborhood shapes. An important fact to be noted here is that computationally, the neighborhood shapes do not necessarily cost more. In our case with Algo IV(r) the cost of computing over a neighborhood is the same as that for neighborhoods of generalized Algo I and Algo II when the sizes of the neighborhoods are equal. It was this discovery which lead to the study of the neighborhood structure and the idea that it is the neighborhood shapes which determine the efficience of stochastic algorithms. This is even more important when it is considered that practically all known heuristic algorithms, or near optimizing algorithms, whether there is a stochastic element to them or not, can be fit into the stochastic algorithmic structure. The case of a deterministic, near optimizing algorithm can usually be looked at as a Stochastic Algorithm of sample size 1, perhaps using a probability distribution which places all weight on 1 or perhaps a few points in $\mathcal{X}$.

Learning Experiments With the Optimal Sequencing Problem

It has been conjectured that important information accumulates during the process of a stochastic algorithm computation. That is, over a period of time, certain areas of the spare $\mathcal{X}$ are likely to be explored more frequently than other areas of $\mathcal{X}$. This might lead a researcher to believe that the area where most of the search is going on is the most promising area to look for the global optimal. If so, this could be taken advantage of by deliberately and perhaps exhaustively carrying on the computation in the most promising places. This could be accomplished by altering the probability function P(x) as the computation proceeds. With this notion in mind, some experiments were set up using the optimal sequencing problem. This consisted of keeping a frequency count of the appearance of all possible links between pairs of points in the local optimal permutations. After some computation, greater probability was placed on those links which appeared to be most popular in the sample of observed local optimal solutions. Also, greater emphasis was placed on those links which occurred when f(p) was relatively small. The net result of these algorithmic experiments (they were called learning algorithms) when performed on data very familiar to us was:

1) The frequency with which the learning algorithm found the optimal (or the better) sequences did not vary significantly from the original AlgoIV(r) algorithms.

2) There was a slight improvement (about 2-3 percent on the average) in the computer time expended to observe a local optimal sequence.

Variation in Algo IV(r)

Another variation in the optimal sequencing study was carried out. In the AlgoIV(r) series, a parameter describing the length of displaced subsequences is used. It starts with the value 1 and increases to the value r and then goes back to 1, etc. A variation in this was attempted, which started the parameter at value r and decreased it to 1 and then back to r, etc. It turned out that this change improved performance on some problems, both in computer time and in finding better sequences. The improvements, when they occurred, were very slight but statistically significant. For most problems, however, there was no significant change in performance.

The optimal sequencing problem aroused much interest in shortest path problems and investigation into generalized shortest path problems was made. Two methods were studied and successfully programmed. One method was especially interesting. It turned out that it was almost as economical, computationally, to calculate the shortest path between all possible pairs of points as it was to calculate the shortest path between a particular pair of points. This was the program developed on the basis of Shimbel's paper [11]. A brief description of the two programs follows. Detailed program listings and operating instructions are available from the author.

MINIMUM PATH THROUGH A NETWORK

This program finds the minimum distance between all pairs from a given set of points or cities or nodes. In general, this is a fairly simple technique; however, it was desired to develop a procedure that can handle very large problems. The basic solution technique is closely related to that described in a paper by A. Shimbel.(11)

The method starts at the base node and adds the distance to the closest node. The method spreads from this base, each time adding the closest node to the chains being developed. When all relevant nodes have been added to the chains, the procedure is finished.

At any given point in the solution, the algorithm assumes there are say N nodes in the network and the optimal distances between these nodes are known. The addition of the (N + 1) node selected as the closest one of the remaining nodes (i.e. outside the N) connected by a single link to one of the N nodes. The node selected from the N nodes is called the established node. Since there are N nodes, this means that at a maximum there are N possibilities for adding in the next node. For each of these possibilities the sum of the distance from the base node to the established node and from the established node to the new node is computed. The next addition to the network is that node with the smallest such total value. This process is repeated until all nodes in the original problem have been added to the network.

For this problem two types of nodes were defined - primary and secondary. The primary nodes are the actual points under consideration, while the secondary nodes are structural nodes necessary for the complete definition of the network. A primary link is the basic connection between a pair of nodes specified by the original problem.

Three types of analysis may be performed with the routine as it is currently set up:

1. All nodes (primary and secondary) are listed along with every node connected by a primary link to them. This enables checking of the input data.

2. All the minimum distances between a specified primary node and all other primary nodes are generated, along with sufficient information for tracing the actual routes.

3. All the minimum distances between all primary nodes are generated.

Another option can be easily added, that is consideration of non-symetric distances. For example, suppose that the network under consideration is an airline transportation network and the primary links show the time for traveling between points. Due to outside factors, such as prevailing winds, the time from A to B may not be the same as from B to A. With minor modifications, this program can be altered to handle this non-symetric case. Also one-way links can be considered, for example, for the case where there is no return B to A.

The program is designed to handle large problems. As it is currently set up it can handle up to 3000 total nodes (primary and secondary) and 8000 total primary links. This is accomplished through packing multiple units of information in each storage location in many cases. The internal storage is dynamically assigned in that there is no limit on the number of links per node as long as the total number of links to all nodes does not exceed 8000. By using this dynamic storage the inefficiency of having unused positions can be avoided. Also, either the number of links or nodes possible can be increased by decreasing the other.

The program has three major parts. Part I is a translator that reads definitions for all nodes (both primary and secondary) and uses these definitions to check and recode raw input data. Part II is a standard sort which groups all information in ascending order by node. Part III performs the actual solution using the method described above, and if the third type of analysis is specified, uses repeated applications of the method to solve for all primary nodes.

As an example, a problem was run for the analysis of 321 cities (i.e. 321 primary nodes) and 328 secondary nodes. There were over 1100 one-way primary links (i.e. over 2200 total links). Analysis time was slightly over 1.5 minutes for each primary node, i.e. after the input phase a type 2 analysis takes about 1.5 minutes plus time required for output.

A Second Computer Algorithm for Finding The Minimum Path Through a Network (MINNET)

The MINNET system is designed to find the route of minimum length through a linked group of elements. The fundamental problem is basically simple. This system uses the procedure described by Hillier and Leberman [12] .

The algorithm begins with the specification of a base node. At any point in the solution N nodes have been added to the network and the distance to each of these nodes is known. For the addition of the N + 1 node, it must be the closest one of the remaining set connected by a single link to the established nodes. Since there are N known nodes, there is a maximum of N possibilities for the next addition. For each of these possibilities the sum of the distance from the base to the established node and from the established node to the new node is computed. The next addition to the network is that node with the smallest total value. The process is repeated until all required nodes have been added to the system.

For small problems there is no problem in the implementation of this type of algorithm. However, to allow for the solution of large problems without partitioning the network, special methods of storing, packing, and referencing have been developed. The problem can handel up to 3000 nodes and up to 5000 links in its current configuration. Through the use of dynamic relative storage assignment and addressing there is no limit to the number of links per node, so long as the total number of links does not exceed 5000. The system is currently set up to run in an 86K environment. The restrictions on nodes and links can, of course, be expanded if more core is available. The system will handel both symetric and non-symetric networks. Non-symetric networks are frequently encountered, especially when the link values are expressed in time. For example, due to some outside factor such as prevailing winds, the time to go from A to B is not equal to the time to go from B to A. This ability to handel non-symetric cases also provides the ability to handle one-way links between nodes. All of these variables in the program are composed of integer half-words (2 bytes). The method sets relative pointers to groups of nodes in a dynamic array and uses multiple levels of indirect addressing. The MINNET System is composed of three phases. The first phase (TRAN) translates the raw input data and generates two files - LINKNET1 and NAMEFL. The second phase (sort) is a sort of LINKNET1. The third and final phase (MIN) generates the minimum paths through the network and uses as input the two files generated in phase 1. Descriptions of each phase showing what the phase does, the files processed and generated, the raw network definition data, and all control cards are available from the author.

The first phase in the solution procedure is the processing of the elementary link cards. Since any code between +5000 and -5000 is valid, a translation table must be set up which defines those nodes which are to be used as part of the network and the internal code of each node. The raw codes may be specified in either the primary or secondary file. As each new code is specified it is assigned a sequential internal code. This internal code begins at 1 and is incremented by +1 for each new code.

The primary file is designed for the specifications of the most important nodes on a node by node basis. For each node a short description may also be included. The secondary file is designed for the specification of groups of less important nodes. An individual node or a group of nodes may be defined on each specification card. No descriptive information is processed.

The last part of the secondary file should be used to describe the structural nodes. These are the nodes that are only defined in order to properly define the network, i.e. a node caused by the intersection of two or more links. This type of node should be assigned the highest internal sequence number, since it is possible to avoid linking all of them (eg. only the ones necessary) in some types of analysis which results in a faster solution. Either the primary or secondary file may be omitted by inserting in it's place a NULLFILE card. An error will occur if an attempt is made to re-define a previously defined node.

The linkage file definition card is used to specify the format of the primary linkage data, define the location of the origin and destination indices, the flag which specifies whether or not the link is symetric or one way, and the calculations that are to be performed in the calculation of the link values.

Up to 10 calculations may be specified on each link card. The calculations may use input variables or immediate data specified in the operation definition. All immediate and input data must be integer.

A through error checking routine is included which provides a complete analysis of any input errors.

The Second Phase in the procedure is merely a sort of a file generated in Phase I.

The third phase of the system generates the actual paths through the network. Several options are available. A range of nodes is defined and each of the nodes in this range is used as the base node in a solution pass. Note - the nodes are referenced by their converted sequence numbers. It is also possible to define at what point the structural, or non-vital nodes begin. Several solution strategies are available which use only those structural nodes necessary for the generation of the minimum paths or trees. This method reduces the solution time and the volume of the output over the case where all nodes in the system are linked into the trees. If the network is symetric, i.e. A => B = B => A, a solution strategy may be used which will reduce the execution time even further.

Input files are automatically rewound before they are read. Therefore, it is possible to use the same data for several applications, each time varying some parameter, e.g. base node, total number of nodes, etc.

The master output file is rewound only on command. Therefore, it is possible to stack several solutions on the same file. An automatic checkpoint feature is included and may be invoked if required. This feature will automatically store all relevant information on a tape at key points during the solution process. If the run is terminated for some reason the restore/restart tape may be used to restart the problem at the last recorded checkpoint.

An Experimental Study of Algorithmic Performance as a Function of Neighborhood Size and Shape

In order to further confirm some conclusions drawn from the optimal sequencing problem, a study was designed using a grid of discrete points in 3 and 4 space. The objective function was an exponential function defined by;

$$f(x,y) = \sum_{i=1}^{n} W_i^2 \, e^{-wi}$$

where $wi = (x-ai)^2/bi^2 + (y-ei)^2/di^2 - 1$

where the a_i, b_i, c_i and d_i are selected constants and x and y varied between -10 and +10 in intervals of $\Delta = .001$. The function which appears in [8] can be controlled to possess as many local maxima as desired by the selection of the parameters a_i, b_i, c_i, d_i and n.

Another experiment was made with a similar function but in 4 dimensions. The function was:

$$f(x,y,z) = \sum_{i=1}^{n} Wi^2 \, e^{-wi}$$

$$\text{where } Wi = (x-ai)^2/bi^2 + (y-ci)^2/di^2 + (z-ei)^2/fi^2 - 1$$

These functions were chosen because of the ease with which the sizes and shapes of the neighborhood structure could be chosen. Two types of neighborhoods were defined which differed primarily by shape. One shape is referred to as square. Its neighborhoods are defined as follows:

The neighborhood of a given point (x,y) consists of all points of the form $(x + e, y + \delta)$ $(x + G, y + \delta)$ where $-r\Delta \leq E \leq r\Delta$ and $-r\Delta \leq \delta \leq r\Delta$ and where r is a parameter which determines the size of the neighborhood. In our experiment r took on value 1, 2, 3, 4 and 5. (See charts on next pages.)

The other class of algorithms were related to the following neighborhood structure and are referred to as star neighborhoods.

The neighborhood of a given point (x,y) consists of the point itself plus all points of the form $(x + \delta, y)$ or $(x, y = \delta)$ where $-(2r + 1)\Delta \leq \delta \leq + (2r+1)\Delta$

The definition of the neighborhoods for functions defined over points in three space were made by the obvious generalization. For these experiments r took on values of 1 and 2. Some results are given in the following table.

Function 1, 2 Dimensions

r.	Neighborhood	Relative frequency global optimal observed	Average number of neighborhood computed
1	Square	.132	396.1
1	Star	.156	304.4
2			
2			

The comparisons, which can be easily made by glancing at the data, give overwhelming evidence of the algrithmic performance differences due to simple changes in neighborhood shapes. In almost all cases, both types of neighborhood structures led to observations of the global optimal observation. In fact, 20 times out of 30 the star neighborhoods outperformed the square neighborhoods. The most significant result here, however, is the overwhelming superiority of the star neighborhoods when comparing the amount of computing involved. The ratios run from approximately 4:3 for r equal to 1 to 4:1 for r equal to 5. Since these ratios are directly related to the computer time involved, the potential savings in computing would be quite substantial. Although this function is not combinational and is rather simple in form, it does serve to bring out the point in the experiment thus re-inforcing the generality of the conclusions drawn from the optimal sequencing studies.

ANALYSIS OF TWO DIMENSIONAL OPTIMIZATION EXPERIMENT

Experiment No. 1

Neighborhood	r-Parameter	Average No. of Steps to local optimal	Relative Frequency Observed Global Maximum
Square	1	396.10	0.132
Star	1	304.43	0.156
Square	2	209.08	0.130
Star	2	102.39	0.160
Square	3	141.89	0.134
Star	3	52.84	0.162
Square	4	104.12	0.096
Star	4	31.88	0.164
Square	5	81.78	0.138
Star	5	22.83	0.152

Experiment No. 2

Neighborhood	r-Parameter	Average No. of Steps to local optimal	Relative Frequency Observed Global Maximum
Square	1	402.40	0.098
Star	1	317.76	0.108
Square	2	193.71	0.126
Star	2	104.40	0.100
Square	3	129.56	0.097
Star	3	49.98	0.116
Square	4	99.87	0.105
Star	4	31.60	0.120
Square	5	80.95	0.124
Star	5	22.21	0.129

Experiment No. 3

Neighborhood	r-Parameter	Average No. of Steps to local optimal	Relative Frequency Observed Global Maximum
Square	2	173.25	0.134
Star	2	90.04	0.120
Square	3	120.88	0.092
Star	3	49.20	0.140
Square	4	82.03	0.102
Star	4	30.81	0.090
Square	5	72.35	0.104
Star	5	23.14	0.112

Experiment No. 4

Neighborhood	r-Parameter	Average No. of Steps to local optimal	Relative Frequency Observed Global Maximum
Square	1	390.41	0.045
Star	1	284.20	0.055
Square	2	205.40	0.080
Star	2	100.20	0.065
Square	3	144.67	0.035
Star	3	52.39	0.095
Square	4	108.23	0.055
Star	4	32.59	0.070
Square	5	85.23	0.065
Star	5	25.71	0.033

Analysis of Two Dimensional Optimization Experiment

Experiment No. 5

Neighborhood	r-Parameter	Average No. of Steps to local optimal	Relative Frequency Observed Global Maximum
Square	1	385.83	0.030
Star	1	269.72	0.050
Square	2	209.38	0.030
Star	2	104.47	0.005
Square	3	130.33	0.0285
Star	3	44.84	0.035
Square	4	92.14	0.070
Star	4	30.13	0.050
Square	5	72.89	0.040
Star	5	21.73	0.050

Experiment No. 6

Neighborhood	r-Parameter	Average No. of Steps to local optimal	Relative Frequency Observed Global Maximum
Square	1	374.47	0.090
Star	1	285.25	0.075
Square	2	201.86	0.075
Star	2	103.99	0.075
Square	3	135.64	0.065
Star	3	55.14	0.050
Square	4	88.64	0.105
Star	4	35.50	0.105
Square	5	78.75	0.075
Star	5	26.08	0.097

Analysis of Three Dimensional Optimization Experiment

Experiment No. 7 (Three Dimensions)

Neighborhood	r-Parameter	Average No. of Steps to local optimal	Relative Frequency Observed Global Maximum
Square	1	647.16	0.104
Star	1	309.40	0.112
Square	2	323.04	0.149
Star	2	99.43	0.150

The Quadratic Integer Programming Problem

Considerable effort was made in attacking the quadratic integer programming problem. In order to obtain some insight into the problem and to accumulate some data for comparative purposes, a comparative study was made of several known algorithms for solving the (regular) quadratic programming problem (9). During the study, a total of 10 problems were developed and computed which were used for computational experiments throughout this portion of the study.

The problem can be stated as:

$$\text{let } f(x_1,\ldots,xn) = \sum_{j=1}^{n} l_j x_j + \sum_{j=1}^{n} \sum_{i=1}^{n} qij\, XiXj$$

$$\text{subject to } \sum_{j=1}^{n} aijXj \le bi \quad i = 1,\ldots,m$$

$$x_j \ge 0 \quad j = 1,\ldots,n$$

x, integer valued j = 1,...,n

The idea from stochastic algorithms is to start with a random feasible point in the convex set of feasible solutions and determine the optimum value in a neighborhood of the initial point. The process is then repeated for the new point. A sequence of locally optimal values is thus obtained.

The problem then is really composed of 2 subproblems:

1) The random choice of a feasible point.

2) Definition of a neighborhood in the feasible set.

An algorithm has been programmed for the IBM 7040 digital computer for finding solutions to the integer quadratic programming problem by using stochastic algorithms.

The procedure for initially selecting a point is based on a method given in [10]. Essentially, an initial point is chosen (not necessarily integer valued); and if it is not feasible, for the non-integer problem, it is reflected across the most distant hyperplane on whose wrong side it lies. The process is repeated and in a finite number of steps, a feasible (non-integer) point is determined. The coordinates are then rounded, and if the resulting integer valued point is not feasible, then a new random starting point is selected and the entire process repeated.

The procedure for defining a neighborhood of a point is to place the feasible point in the center of a square grid of points determined by a given feasible point. That is, the neighborhood of $(x_1^o, x_2^o,\ldots,x_n^o) = x^o$ is the intersection of the set of feasible points with the set

$$\left\{ x_1^o, \ldots x_s, \ldots x_t, \ldots x_n^o) : \begin{array}{l} x_s^o - k \leq x_s \leq x_s^o + k \\ x_t^o - k \leq x_t \leq x_t^o + k \end{array} \right\} \equiv n(x^o)$$

where k is a specified positive integer and s, t are randomly selected components. The point in G, yielding the lowest function value, then becomes the center of a new grid which defines the successor point's neighborhood.

A local optimum results when no further decrease in function value can be made by simultaneously considering any two components and their increments. The process is repeated in its entirety for a pre-determined number of observations according to some stopping rule.

In order to describe the computational scheme for the determination of a local optimum, the following problem formulation is assumed:

minimize

$$z = \sum p_j x_j + \Sigma_i \Sigma_j \ xiqi_j x_j$$

subject to $\sum_{j=1}^{n} ai_j x_j \leq bi \quad i = 1,2,\ldots,m$

$$x_k \geq 0 \quad k = 1,2,\ldots,n \text{ and integer valued.}$$

let $(x_1^o, \ldots, x_r^o, \ldots, x_s^o, \ldots, x_n^o)$ be the initially feasible, randomly chosen point.

If t_1 is added to x_r^o $\quad -k \leq t_1 \leq k \quad$ and t_2 added to x_s^o $\quad -k \leq t_2 \leq k$

then the difference in function values becomes:

$$(p_r^o + 2 \sum_{i=1}^{n} q_{ir} x_i^o)\, t_1 + (p_s^o + 2 \sum_{j=1}^{n} q_{js} x_j^o) t_2 + 2\, q_{rs} t_1 t_2 + q_{rr} t_1^2 + q_{ss} t_2^2 =$$

$$At_1 + Bt_2 + Ct_1 t_2 + Dt_1^2 + Et_2^2$$

which, if negative, shows a decrease in function value. The value of t_i is fixed (starting at $-k$) and bounds are calculated for t_2 by the inequality;

$$a_{is} t_2 \leq b_i - \Sigma a_{ij} x_j^o - a_{ir} t_1 \quad i=1,2,\ldots$$

Calculations are then made for the allowed range of t_2 with the change in function value calculated by the above formula. This is repeated for increasing values of t_1 up to k.

A new point is thus determined in the obvious way which is then the successor to x.

Some results have been obtained for a set of ten 10 variable, 10 constraint problems given in (9).. The results of the computational experiments indicated that their probably isn't too much advantage in making the grid size larger than 3 or 4 for problems in this order of magnitude. There seemed to be a definite increase in computer time used when k reached a value as large as 5. Several times, k=2 was the best performing value. In each case, solutions were found which yielded values of the objective function very close to the optimal values found in the non-discrete problem - usually within 4-5%. For this particular group of problems (which were generated randomly) the performance of our method was best for problems with large functional values. This can be explained by round-off error, however, and has nothing to do with the algorithms' performances. The frequency with which the global optimal (or best known) solution was found varied markedly from one problem to another. This is interesting, but perhaps not completely unexpected with problems of this complexity. The same result occurred with the optimal sequencing experiments. The answer probably lies in the curious complexity of combinatorial problems.

Some examples of the computational results are:

Continuous Solution f value = -32.44	Best Discrete Solution f value = -28.8
x_1 = 1.84	1
x_2 = 2.75	3
x_3 = 1.14	1
x_4 = 0	0
x_5 = .80	1
x_6 = .60	1
x_7 = 4.45	5
x_8 = .90	1
x_9 = .97	1
x_{10} = 3.86	3

for k=5 best value observed 13 out of 19 times
for k=2 best value observed 9 out of 14 times

Continuous Solution f value = -94.75	Best Discrete Solution f value = -91.05
x_1 = 1.08	1
x_2 = 2.48	2
x_3 = 8.11	8
x_4 = 1.69	1
x_5 = 0	0
x_6 = 4.04	4
x_7 = 0	0
x_8 = 2.85	3
x_9 = 2.97	3
x_{10} = 5.96	6

for k=5, best solution observed 8 out of 14
for k=2, best solution observed 11 out of 24

These results convince us that our method is a valuable one for these types of problems. Our difficulty in further establishing this, however, has been the lack of published experimental results by authors working on the same program. Their examples have always been of a trivial (3 or 4 variables) variety.

PROGRAMS FOR SOLVING THE GENERAL QUADRATIC PROGRAMMING PROBLEM

Quadratic Programming by the Method of Dantzig

The algorithm 'OPMIND' solves the quadratic programming problems by the method of Dantzig. The complete mathematical basis for this method is examined in "An Experimental Study of Some Quadratic Programming Algorithms" by John McGraw Rooker, Technical Report Number 7, University Computing Center, The University of Tennessee.

Basically, one of two sets of calculations is performed, depending on whether or not the problem is in standard or non-standard form. The problem is standard if for any variable either the primal or dual element is in the basis, but not both. The problem is in non-standard form if for any variable(s) both the primal and dual elements are in the solution, forcing another variable to have neither element in the basis.

A. Standard Form - The non-basic element with the smallest negative term is chosen to enter. The element to be replaced is the one first driven to zero. After these elements are known the proper transformations are made. If the element driven to zero is the complementry element of the entering element, the problem stays in standard form and the process is repeated, otherwise the problem is in a non-standard form and process B is used.

B. Non-Standard Form - an attempt is made to bring the problem back to standard form. An attempt is made to make basic the dual element of the variable with neither dual or basic elements in the basis. If the element forced out makes the problem standard, use process A, otherwise, repeat process B.

Summary of Subroutines Used:

QPMIND - Control Program. Performs all input and output; controls iterations depending on whether the problem is in a standard or non-standard state. Also, it performs all checks for valid solutions, no solutions, or infeasible solutions.

MINY - Finds the variable not in the basis to be entered - problem must be in standard state.

VAROUT - Finds the variable in the basis to be removed when the entering variable is known - problem must be in standard state.

TRAN – Transforms the matrix given the variable to enter and the one to remove - used in either standard or non-standard state.

YIN – Used to attempt to drive the problem back to the standard state. Given a variable with both primal and dual elements out of basis, it attempts to replace dual element in basis - used when in non-standard state.

FVAL – Given the final levels of all primal elements in the basis FVAL determines the final function value.

Input Matrix:

A. Seven elements per card, ten columns per element (7F10.2). The elements of A are punched row-wise. Figure 1 shows the construction of matrix A.

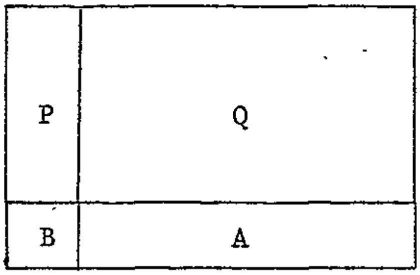


FIGURE 1

where:

P_i - are the linear terms of the quadratic function.
Q_{ij} - if $i = j$ - squared terms of the quadratic function.
if $i \neq j$ - cross-product terms of the quadratic function.
A_{ij} - coefficients of the linear constraints.
B_i - limit on constraint i.

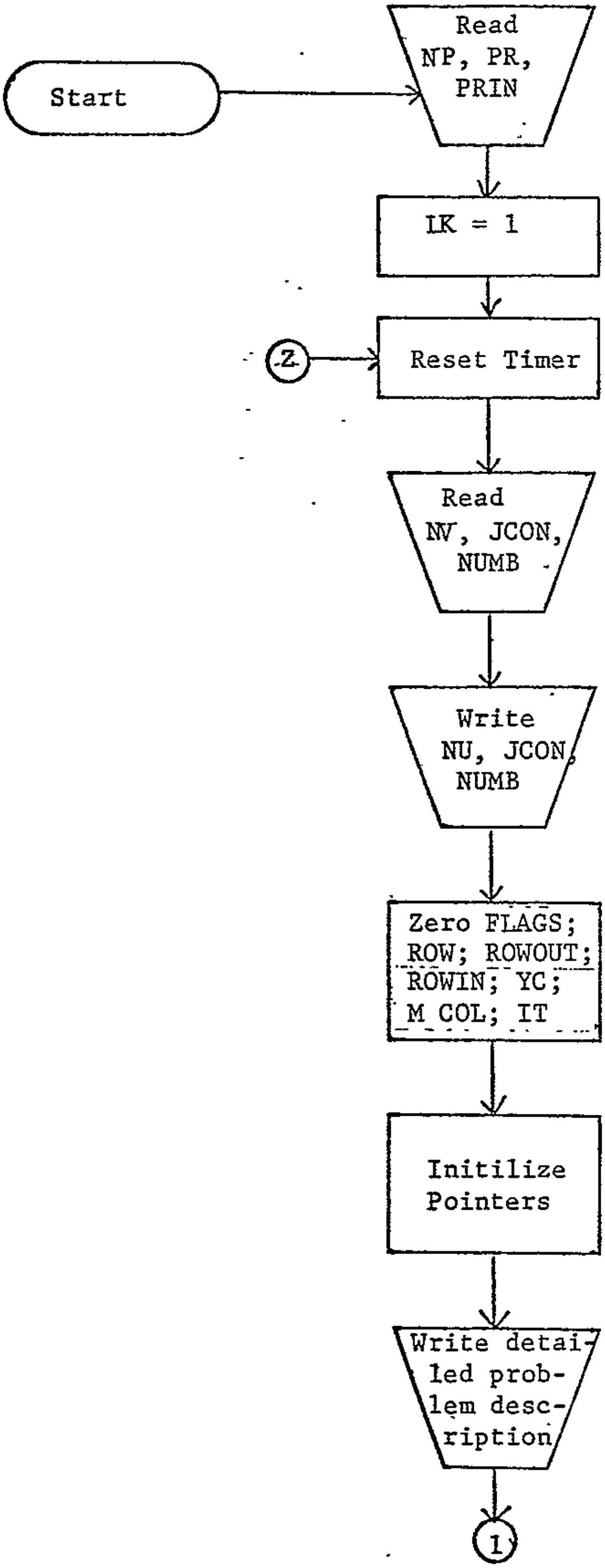
NOTE: ** Only less than or equal type constraints can be used and B_i must be positive, also the number of activities + constraints must be less than 99, and the number of activities must be less than 50.

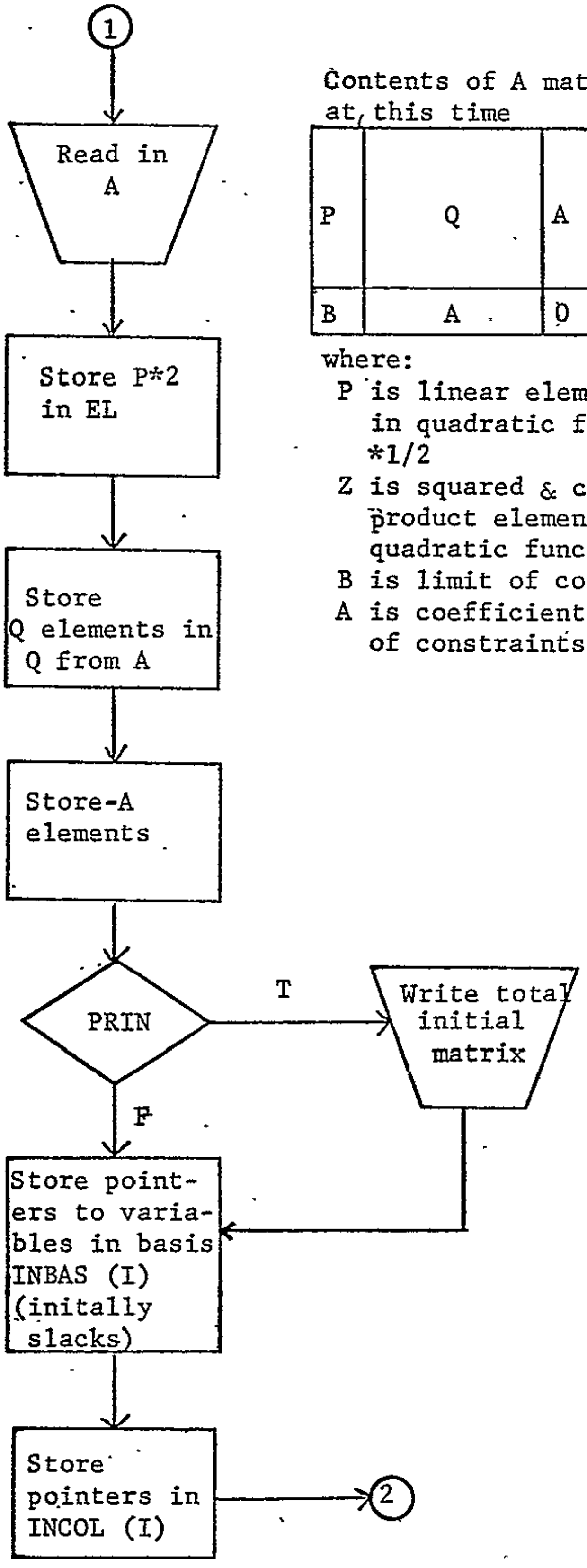
Flow charts for the program follow. Program listings are available from the author.

Program - Qpmind

Quadratic Program

Method of Dantzig

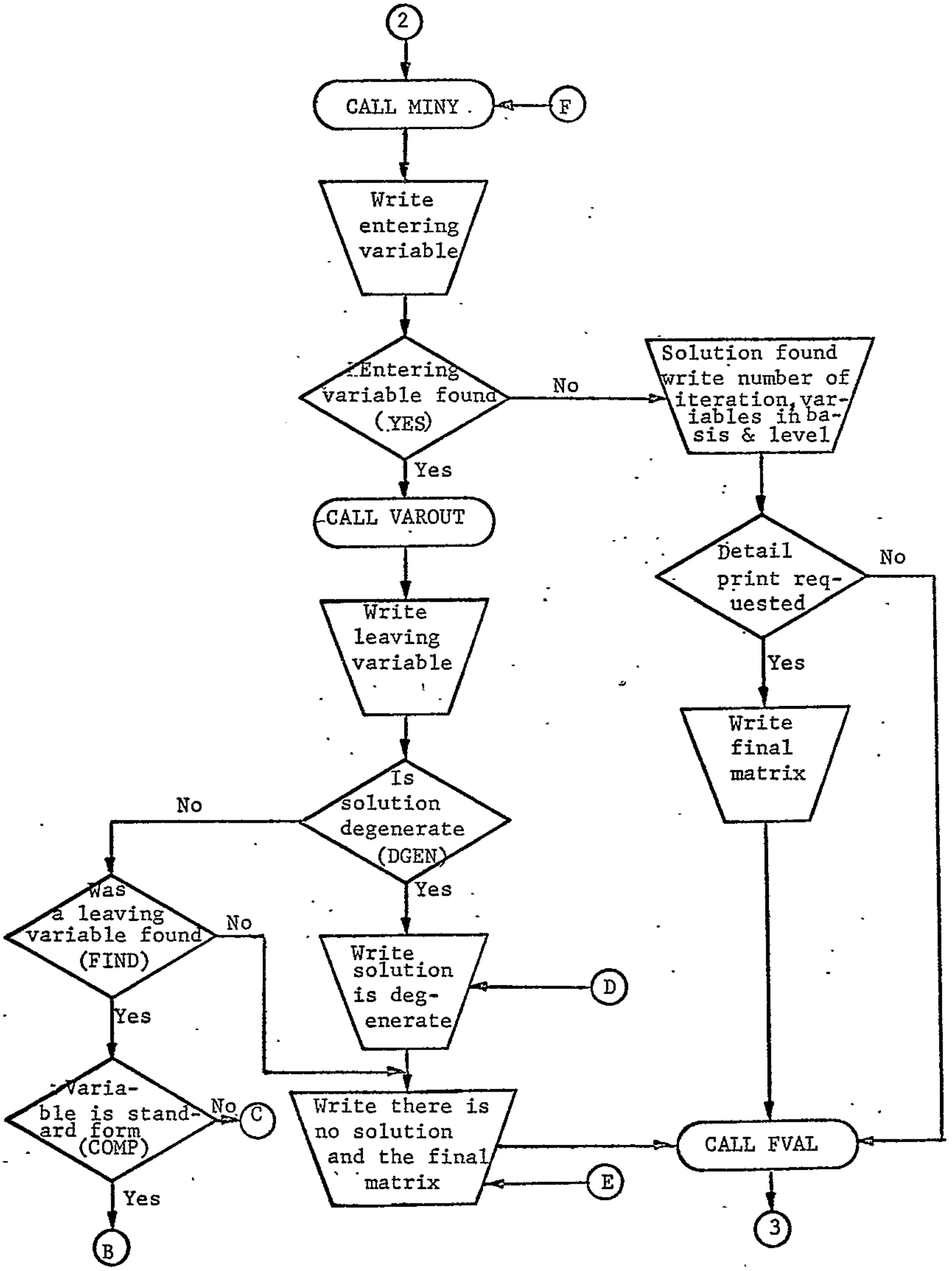


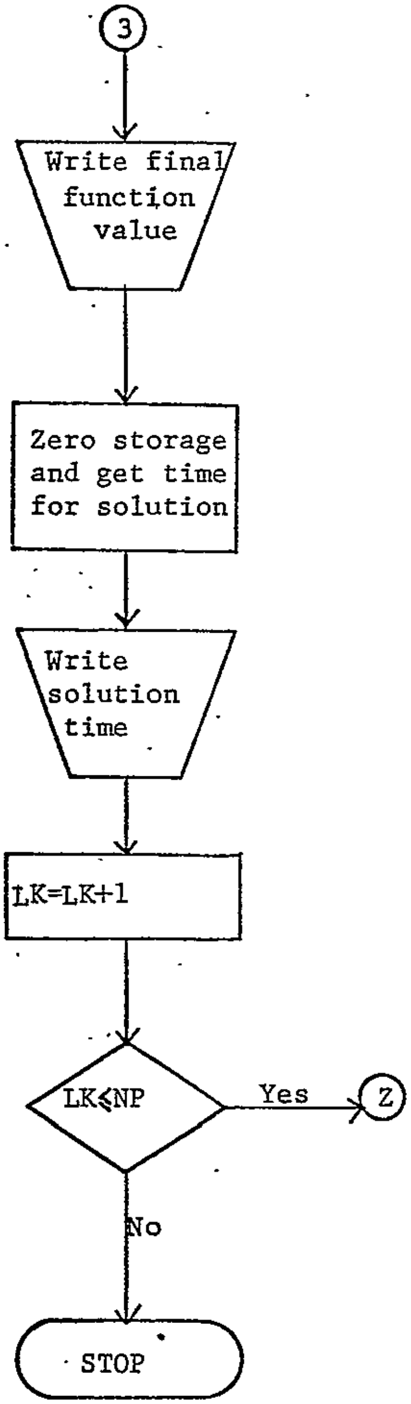

1
Read in A
Store P*2 in EL
Store Q elements in Q from A
Store-A elements
PRIN
T
Write total initial matrix
F
Store pointers to variables in basis INBAS (I) (initally slacks)
Store pointers in INCOL (I)
2

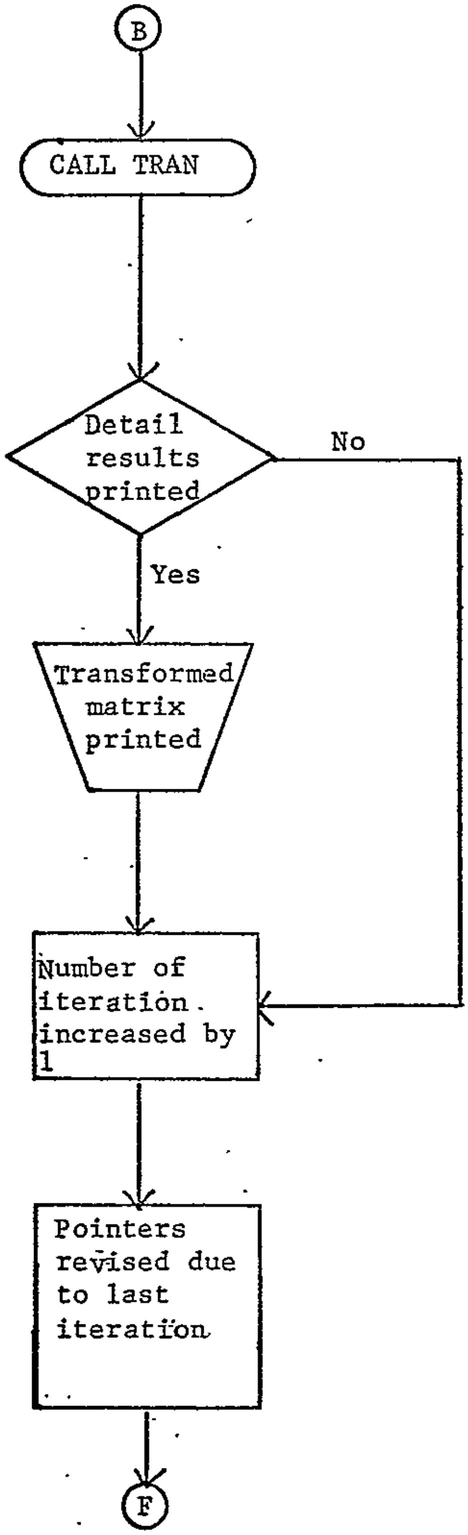
Contents of A matrix at this time

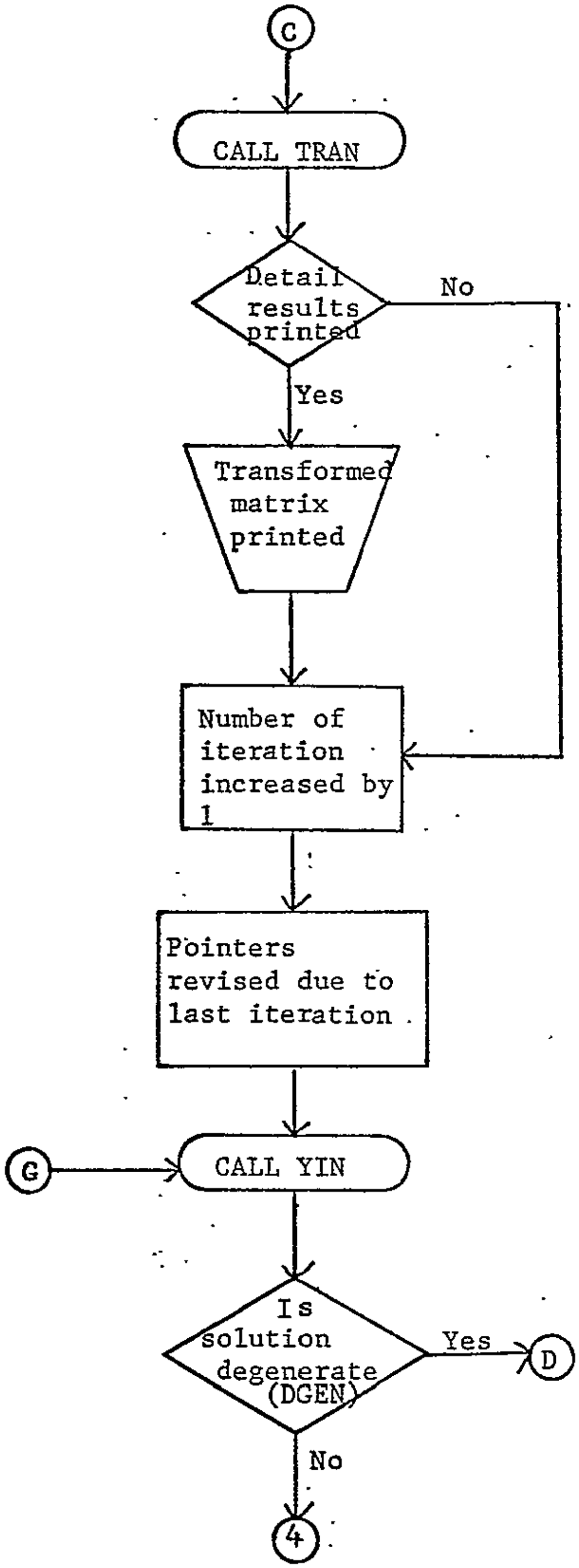
P	Q	A
B	A	0

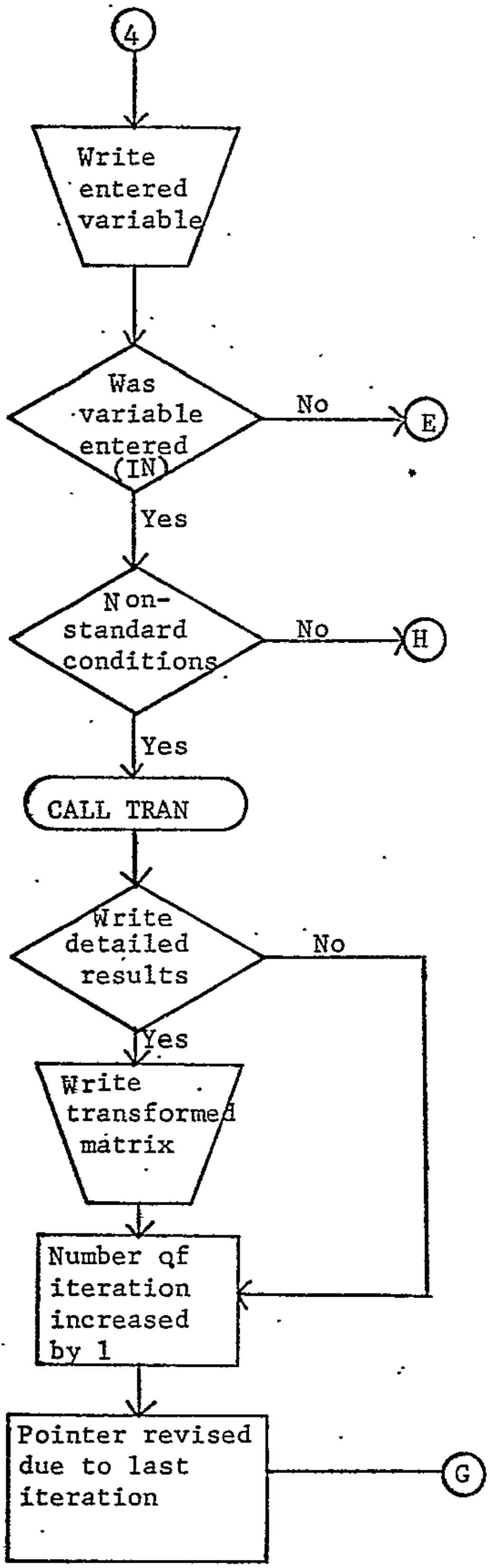
where:
P is linear elements in quadratic function *1/2
Z is squared & cross product elements in quadratic function
B is limit of constraints
A is coefficients of constraints

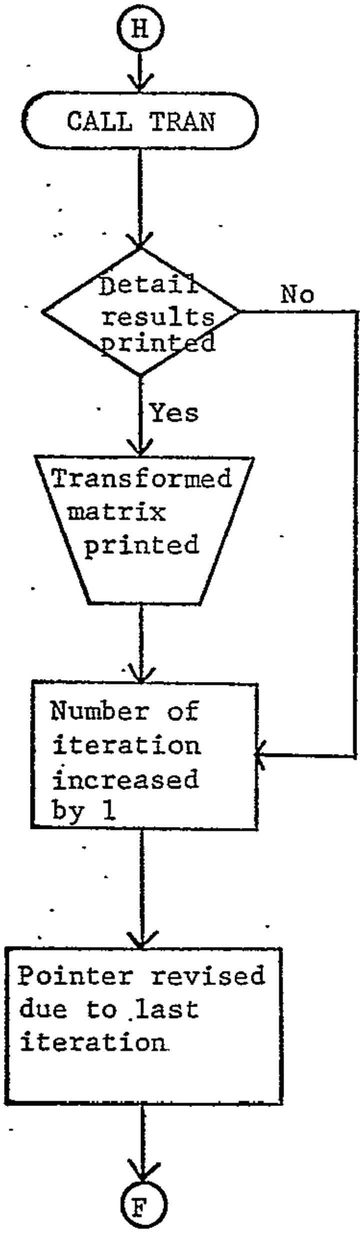

2
CALL MINY
F
Write entering variable
Entering variable found (YES)
No
Solution found write number of iteration, variables in basis & level
Yes
CALL VAROUT
Detail print requested
No
Write leaving variable
Yes
Write final matrix
No
Is solution degenerate (DGEN)
Was a leaving variable found (FIND)
No
Yes
Write solution is degenerate
D
Yes
Variable is standard form (COMP)
No
C
Write there is no solution and the final matrix
CALL FVAL
E
Yes
B
3


3
Write final function value
Zero storage and get time for solution
Write solution time
LK=LK+1
LK≤NP
Yes
Z
No
STOP

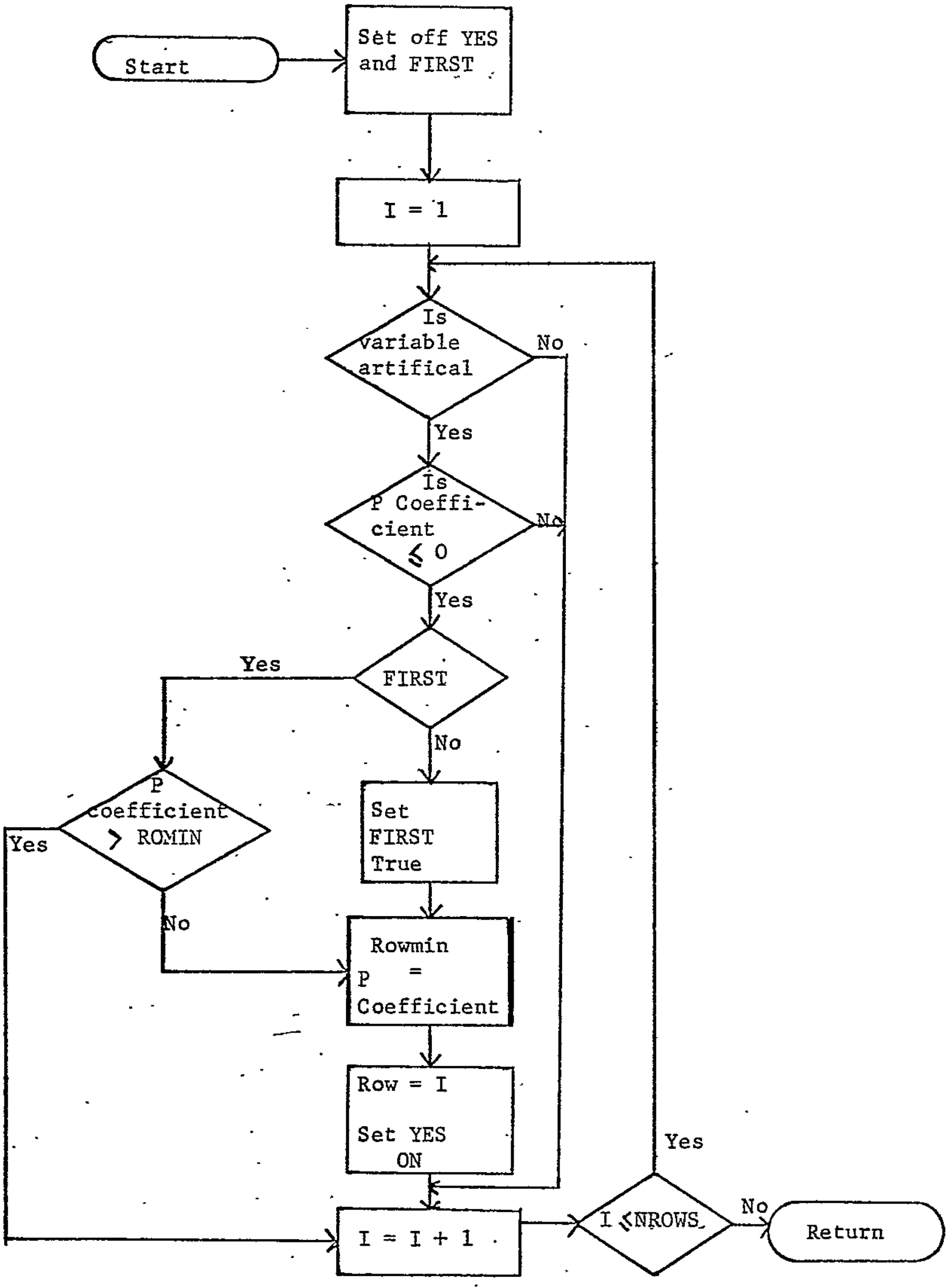

B
CALL TRAN
Detail results printed
No
Yes
Transformed matrix printed
Number of iteration increased by 1
Pointers revised due to last iteration
F


C
CALL TRAN
Detail results printed
No
Yes
Transformed matrix printed
Number of iteration increased by 1
Pointers revised due to last iteration
G
CALL YIN
Is solution degenerate (DGEN)
Yes
D
No
4

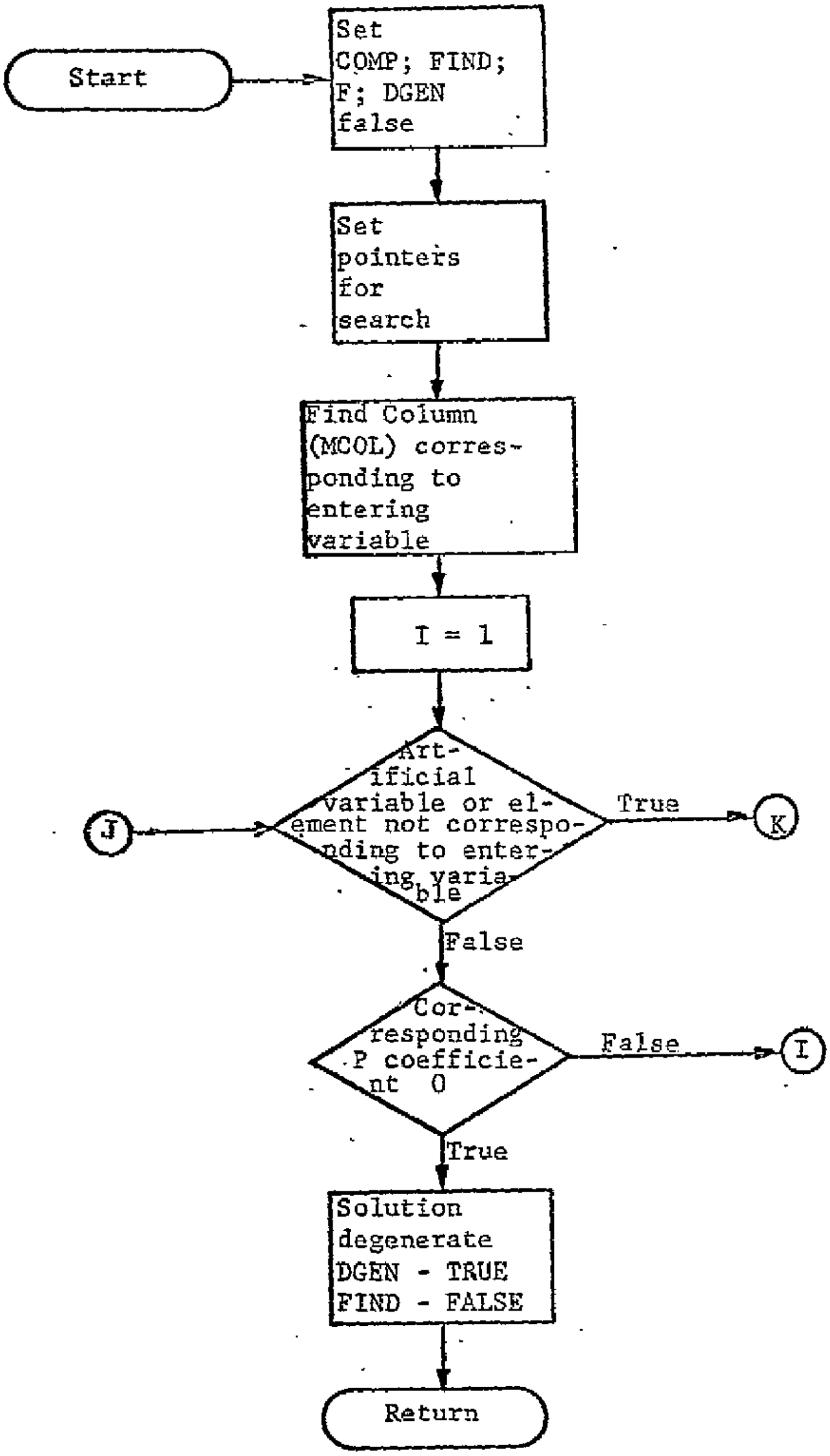

4
Write entered variable
Was variable entered (IN)
No
E
Yes
Non-standard conditions
No
H
Yes
CALL TRAN
Write detailed results
No
Yes
Write transformed matrix
Number of iteration increased by 1
Pointer revised due to last iteration
G


H
CALL TRAN
Detail results printed
No
Yes
Transformed matrix printed
Number of iteration increased by 1
Pointer revised due to last iteration
F

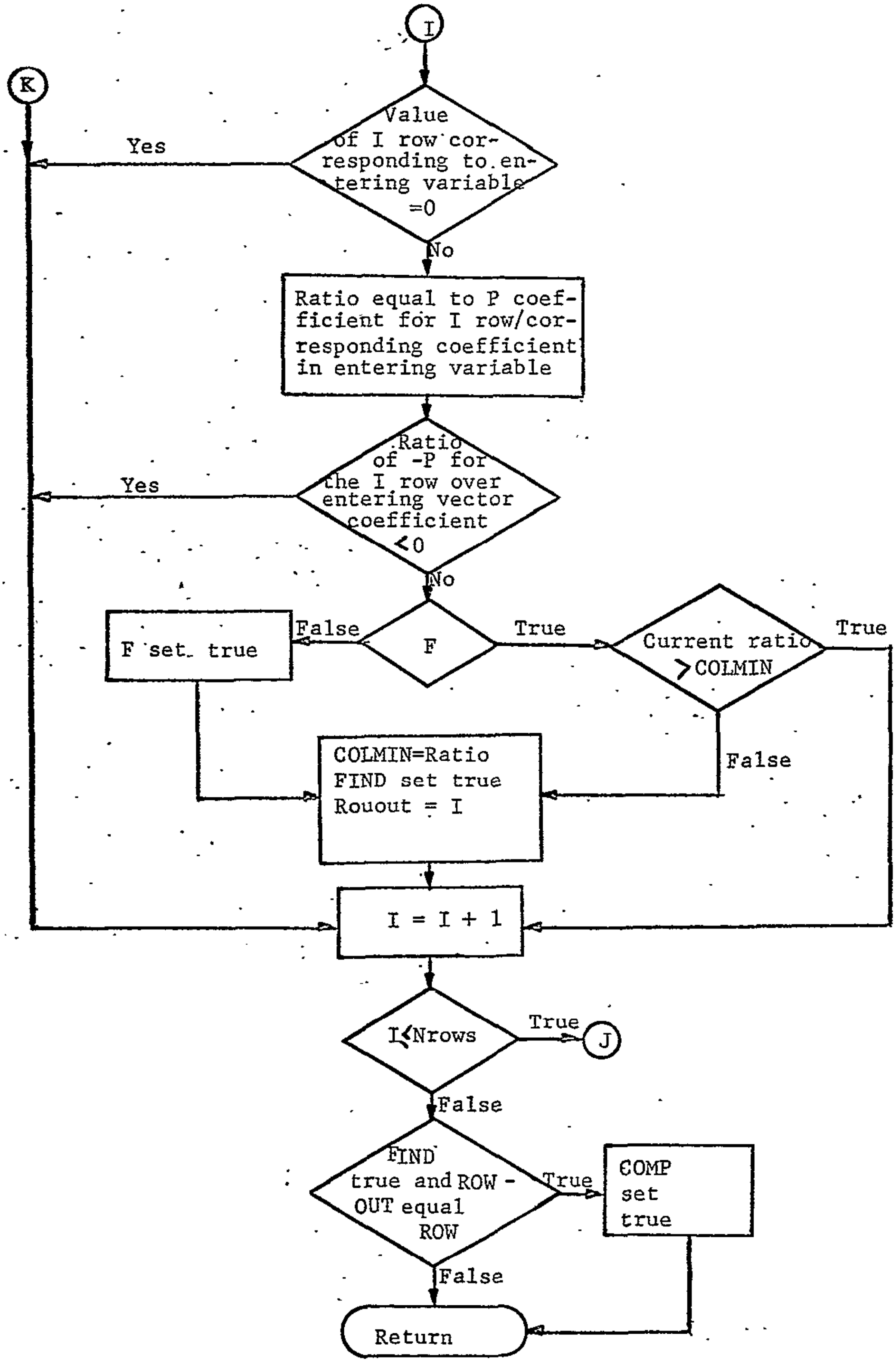
Subroutine MINY



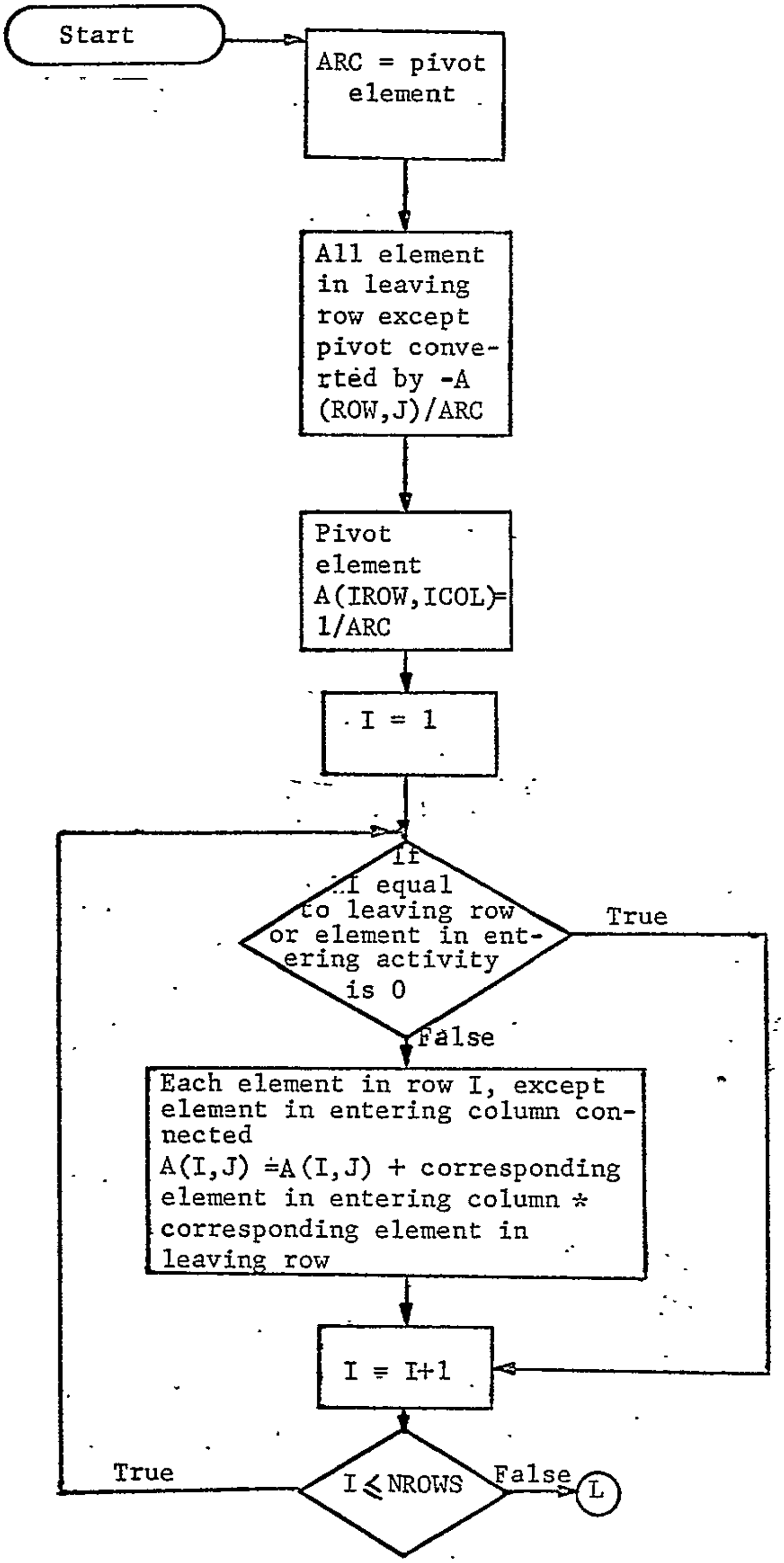
Subroutine VAROUT


Start
Set COMP; FIND; F; DGEN false
Set pointers for search
Find Column (MCOL) corresponding to entering variable
I = 1
J
Artificial variable or element not corresponding to entering variable
True
K
False
Corresponding P coefficient 0
False
I
True
Solution degenerate DGEN - TRUE FIND - FALSE
Return

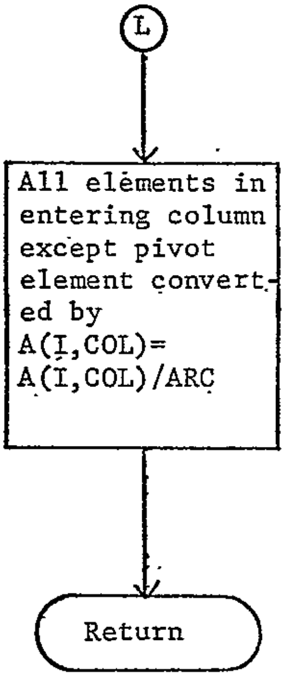
Subroutine. VAROUT



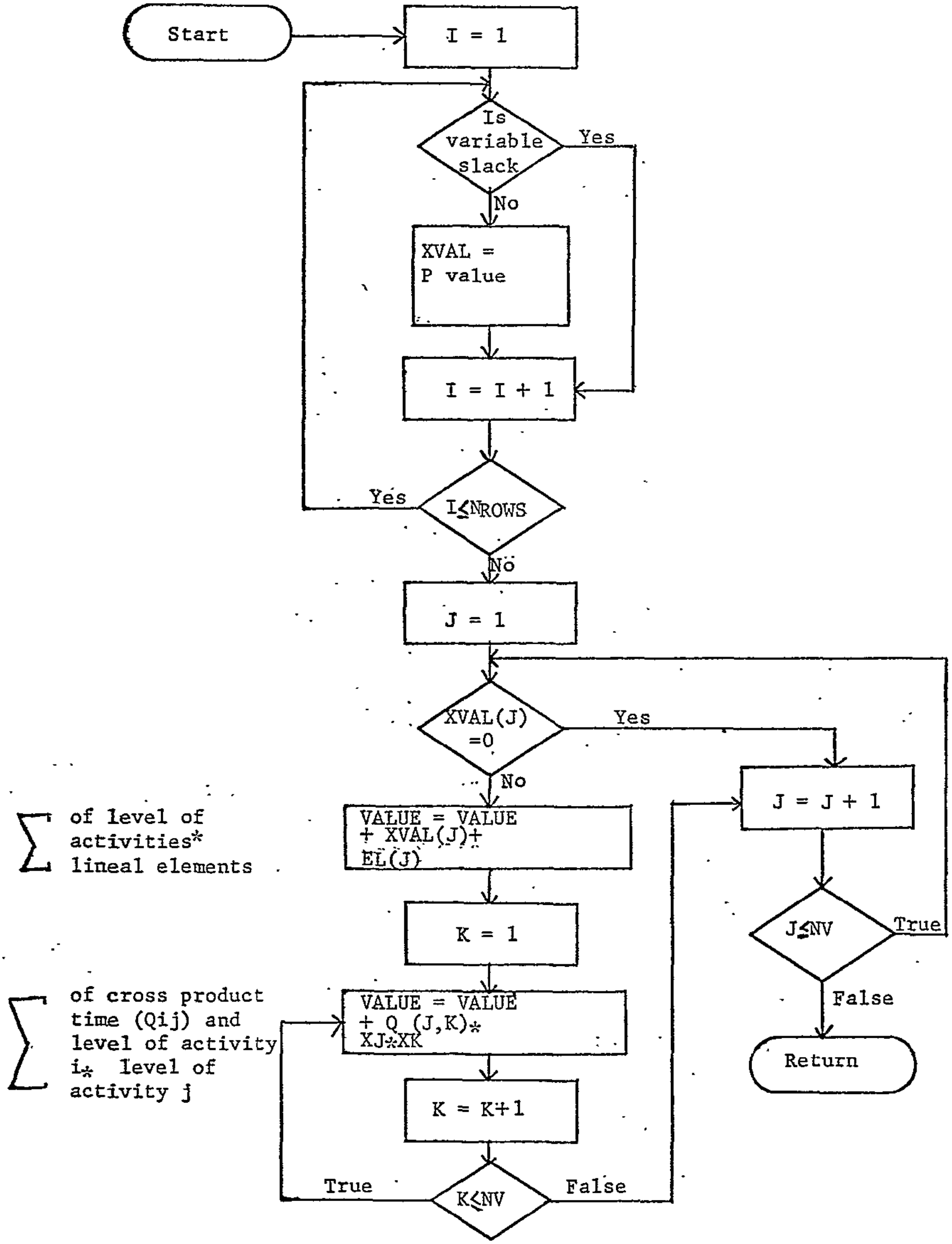
I
K
Value of I row corresponding to entering variable =0
Yes
No
Ratio equal to P coefficient for I row/corresponding coefficient in entering variable
Ratio of -P for the I row over entering vector coefficient <0
Yes
No
F set true
False
F
True
Current ratio >COLMIN
True
False
COLMIN=Ratio
FIND set true
Rouout = I
I = I + 1
I<Nrows
True
J
False
FIND true and ROW - OUT equal ROW
True
COMP set true
False
Return


Start
ARC = pivot element
All element in leaving row except pivot converted by -A (ROW,J)/ARC
Pivot element A(IROW,ICOL)= 1/ARC
I = 1
If I equal to leaving row or element in entering activity is 0
True
False
Each element in row I, except element in entering column connected A(I,J) =A(I,J) + corresponding element in entering column * corresponding element in leaving row
I = I+1
I ≤ NROWS
True
False
L

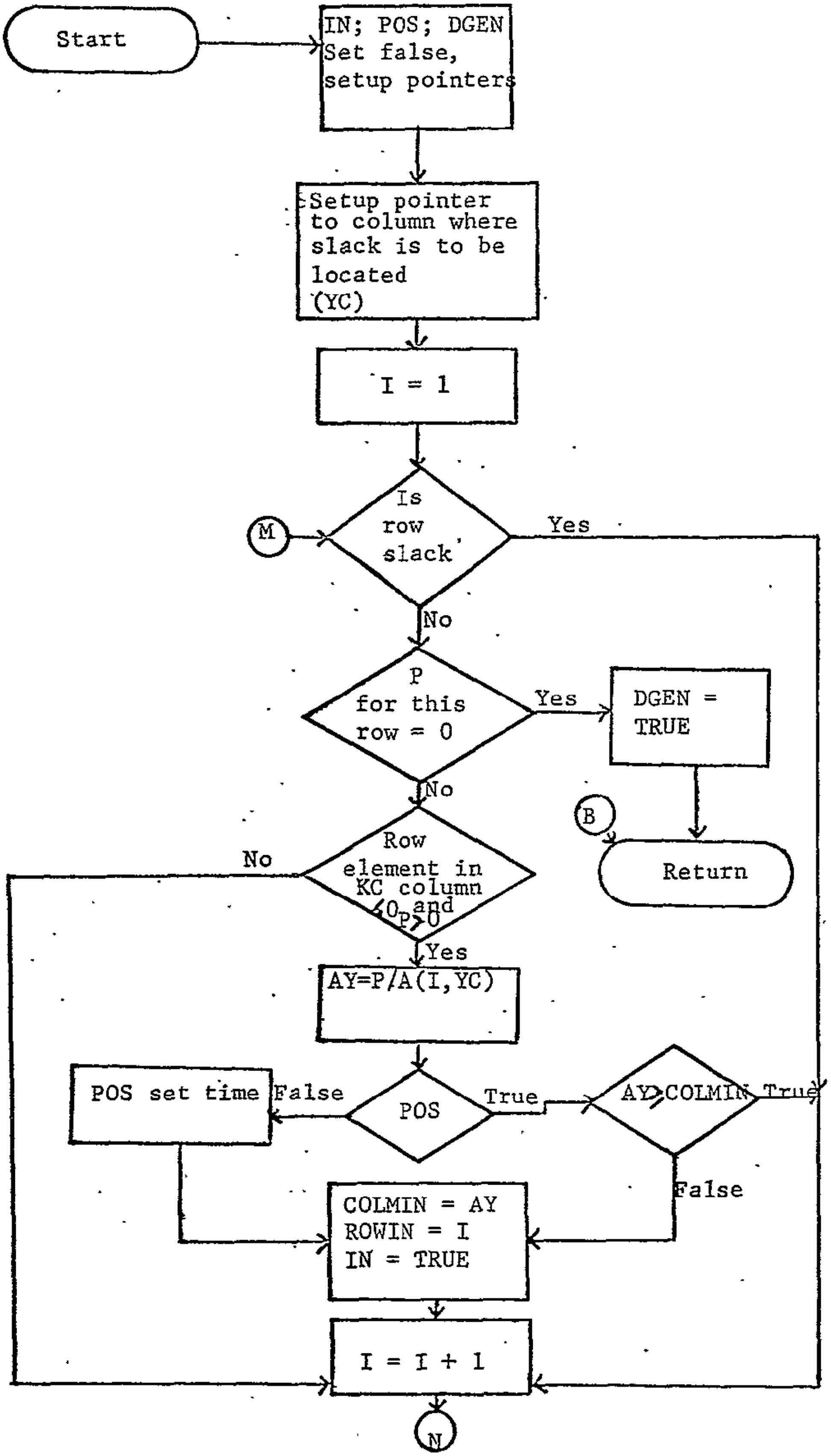
Subroutine TRAN


L
All elements in entering column except pivot element converted by A(I,COL)= A(I,COL)/ARC
Return

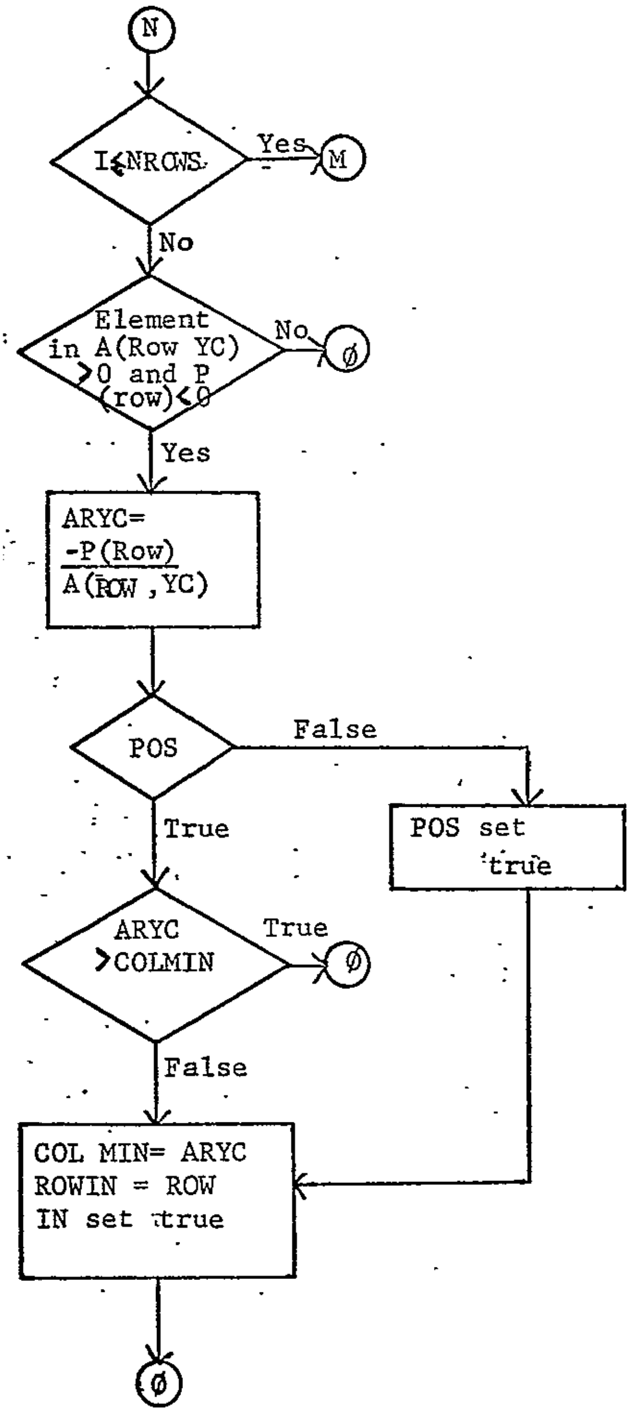
Subroutine FVAL


Start
I = 1
Is variable slack
Yes
No
XVAL = P value
I = I + 1
Yes
I≤NROWS
No
J = 1
XVAL(J) =0
Yes
No
J = J + 1
∑ of level of activities* lineal elements
VALUE = VALUE + XVAL(J)+ EL(J)
J≤NV
True
False
K = 1
∑ of cross product time (Qij) and level of activity i* level of activity j
VALUE = VALUE + Q (J,K)* XJ*XK
Return
K = K+1
True
K≤NV
False

Subroutine YIN



Subroutine YIN



QUADRATIC PROGRAMMING BY THE METHOD OF BEALE

The algorithm 'QPMDPD' solves the quadratic programming problem by the method of Beale. The method is described and illustrated in 'An Experimental Study of Some Quadratic Programming Algorithms' by John McGraw Rooker, Technical Report Number 7, University Computing Center, The University of Tennessee.

The computational scheme utilizes the following rules:

1. Make the transformation according to the component corresponding to max $|pu_i^k|$. That is determine that sign unrestricted component say $x_{u_1}^k$, such that $p_{u_1}^k \neq 0$ and $|p_{u_1}^k| \geq |p_{u_i}^k|$, i = 1, 2, ..., t. Use a suitable rule for breaking ties. Then determine the transformation (Type I or Type II) which leaves all components except $x_{u_1}^k$ fixed. (Type I or Type II) which leaves all components except $x_{u_1}^k$ fixed. (Type I is of the type $x_j \rightarrow c_o + CX$; Type II is $x_j \rightarrow p_j/2 + Q_jX$).
2. If there is no sign restricted component (or if the p^k elements corresponding to sign unrestricted components all vanish), then make the transformation according to the sign restricted component corresponding to max $|p_{r_i}^k|$, $p_{r_i}^k < 0$. (See page 3 for definition of terms.)

Summary of Subroutines Used:

QPMBPD - Control program. Performs all input and output and controls iterations for transformations.

FINMV - Finds the component not in the basis to be entered. Calls SUBROUTINE MAXIND.

MAXIND - Finds the component in the basis to be removed given the entering component. Determines the type transformation to be made.

TRAN - Performs the transformation.

DGCK - Checks for degeneracy of solution.

INFCK - Checks for infeasibility of solution.

Input:

Card No.	Data	Format
LEAD	NP, PR, FIRST	(I5, 4X, L1, 4X, L1)
	NP - number of problems to be solved. PR - print control. T for printing of tableau after each transformation; F otherwise. FIRST - print control. T for printing of first and last tableaus; F otherwise	
	NV, JCON, NUMB	(3I5)

	NV = number of variables - including one slack variable.
	JCON = number of constraints.
	NUMB = arbitrary problem number.	

2	A(I,J) J = 1,NV, I = 1,IROWS where IROWS = NV + JCON	(7F10.2)

A(I,J) - the A matrix is set up as follows:

A(I,J)

$$\begin{array}{llllll}
 & & X_1 & X_2 & & X_{NY} \\
 & 0 & p_1/2 & p_2/2 & \cdots\cdots & p_{NV}/2 \\
 & p_1/2 & q_{11} & q_{12} & \cdots\cdots & q_{1,NV} \\
 & p_2/2 & q_{21} & q_{22} & \cdots\cdots & q_{2,NV} \\
 & \vdots & \vdots & \vdots & & \vdots \\
 & p_{NV}/2 & q_{NV,1} & q_{NV,2} & \cdots\cdots & q_{NV,NV} \\
Y_1 & b_1 & a_{11} & a_{12} & \cdots\cdots & a_{1,NV} \\
Y_2 & b_2 & a_{21} & a_{22} & \cdots\cdots & a_{2,NV} \\
 & \vdots & \vdots & \vdots & & \vdots \\
Y_{JCON} & b_{JCON} & a_{JCON,1} & a_{JCON,2} & \cdots\cdots & a_{JCON,NV}
\end{array}$$

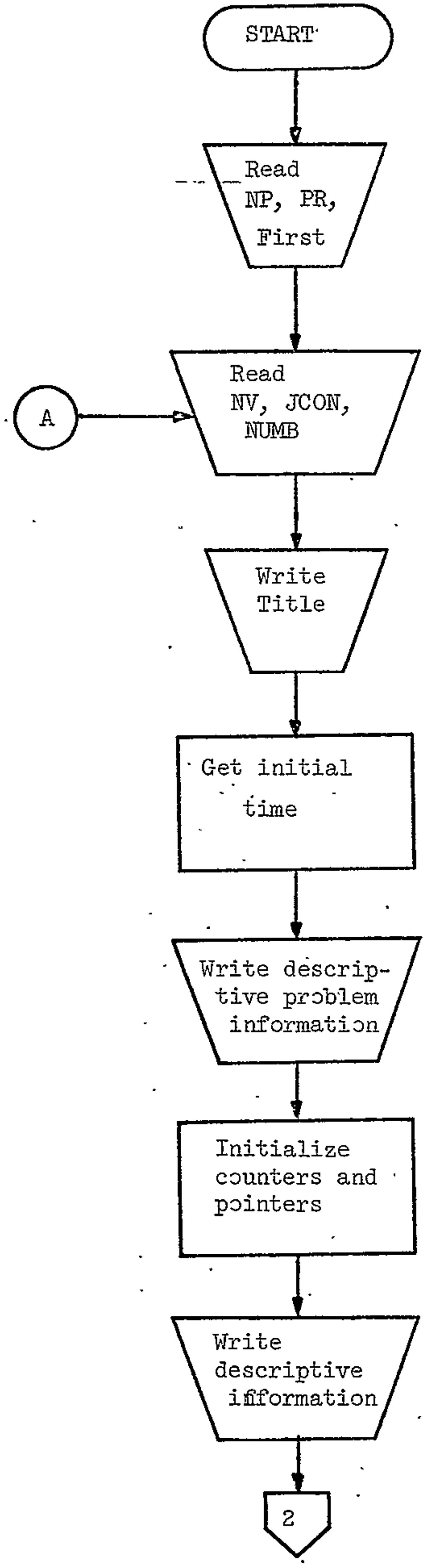
Where p_i are the coefficients of the linear terms of the objective function;

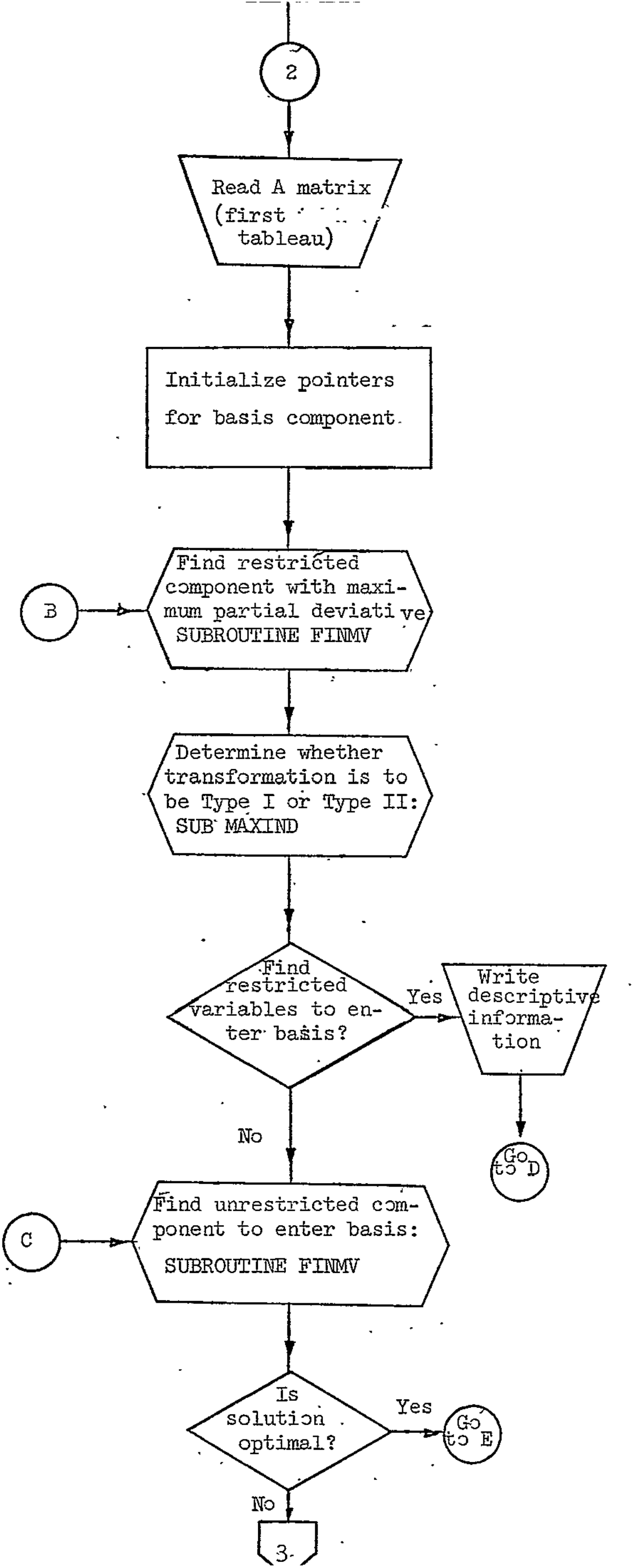
q_{ij} are the coefficients of the cross product terms ($i \neq j$) and the squared terms ($i = j$) of the objective function;

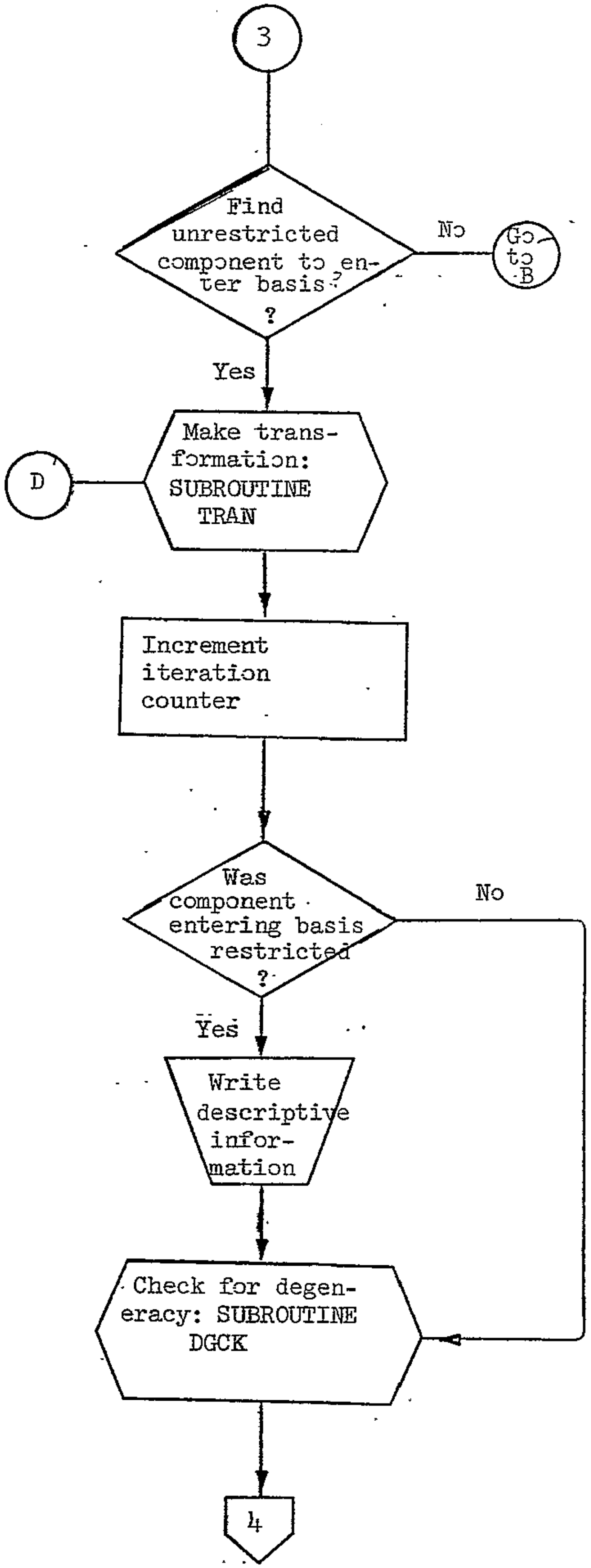
b_i are the constant terms of the constraints; and

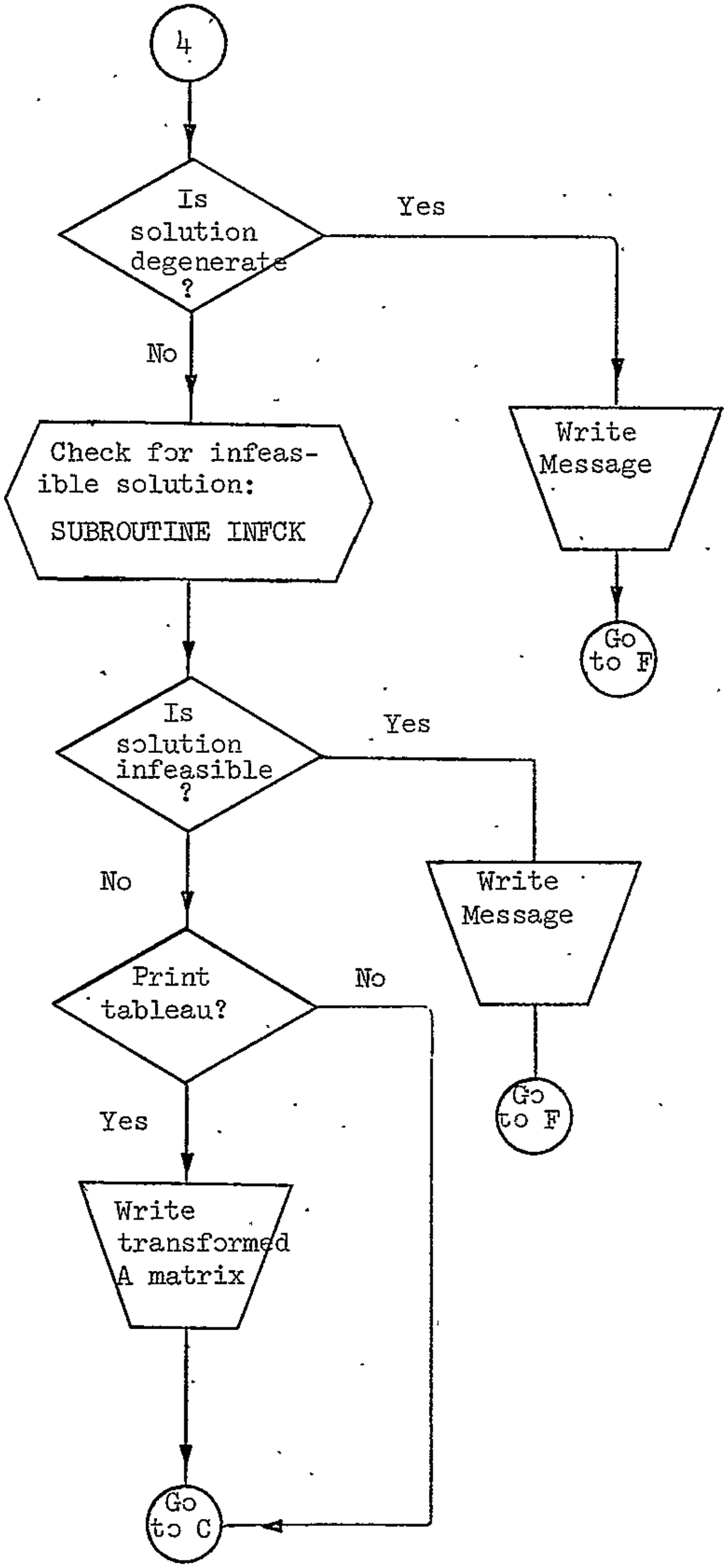
a_{ij} are the negatives of the coefficients of the linear terms of the constraints.

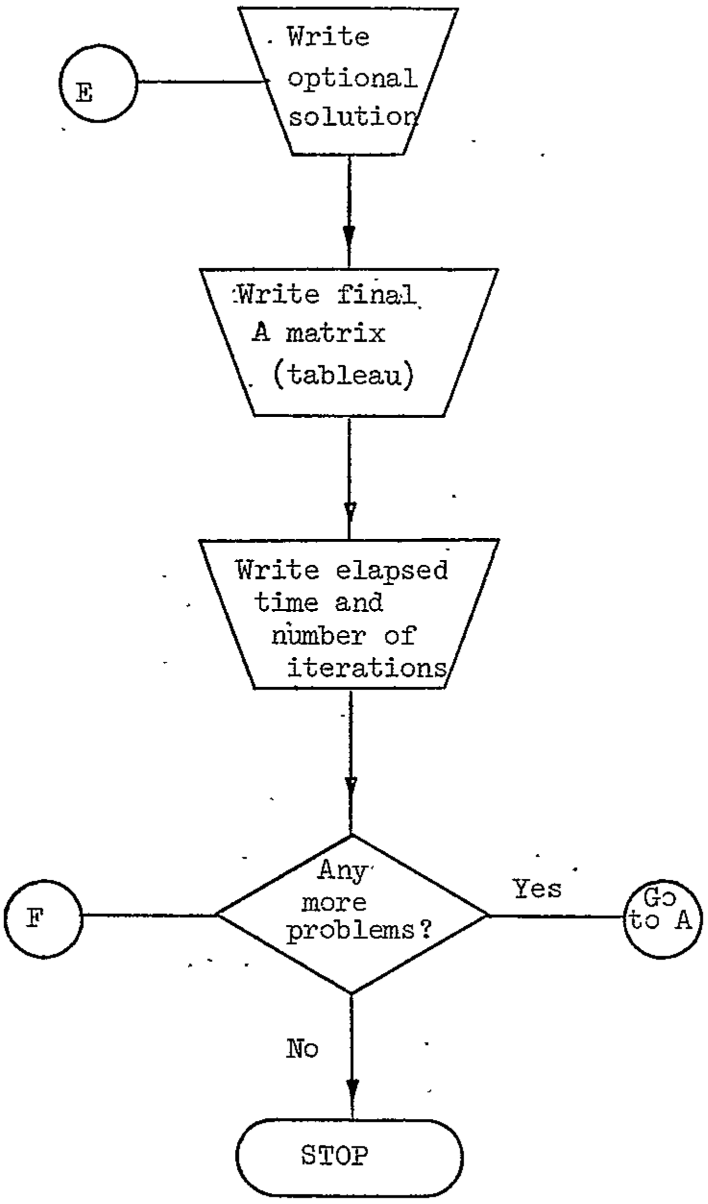
NOTE: Present dimensions restrict the number of rows in the A matrix (tableau) to 150 and the number of columns to 50.


START
Read NP, PR, First
A
Read NV, JCON, NUMB
Write Title
Get initial time
Write descriptive problem information
Initialize counters and pointers
Write descriptive iflformation
2

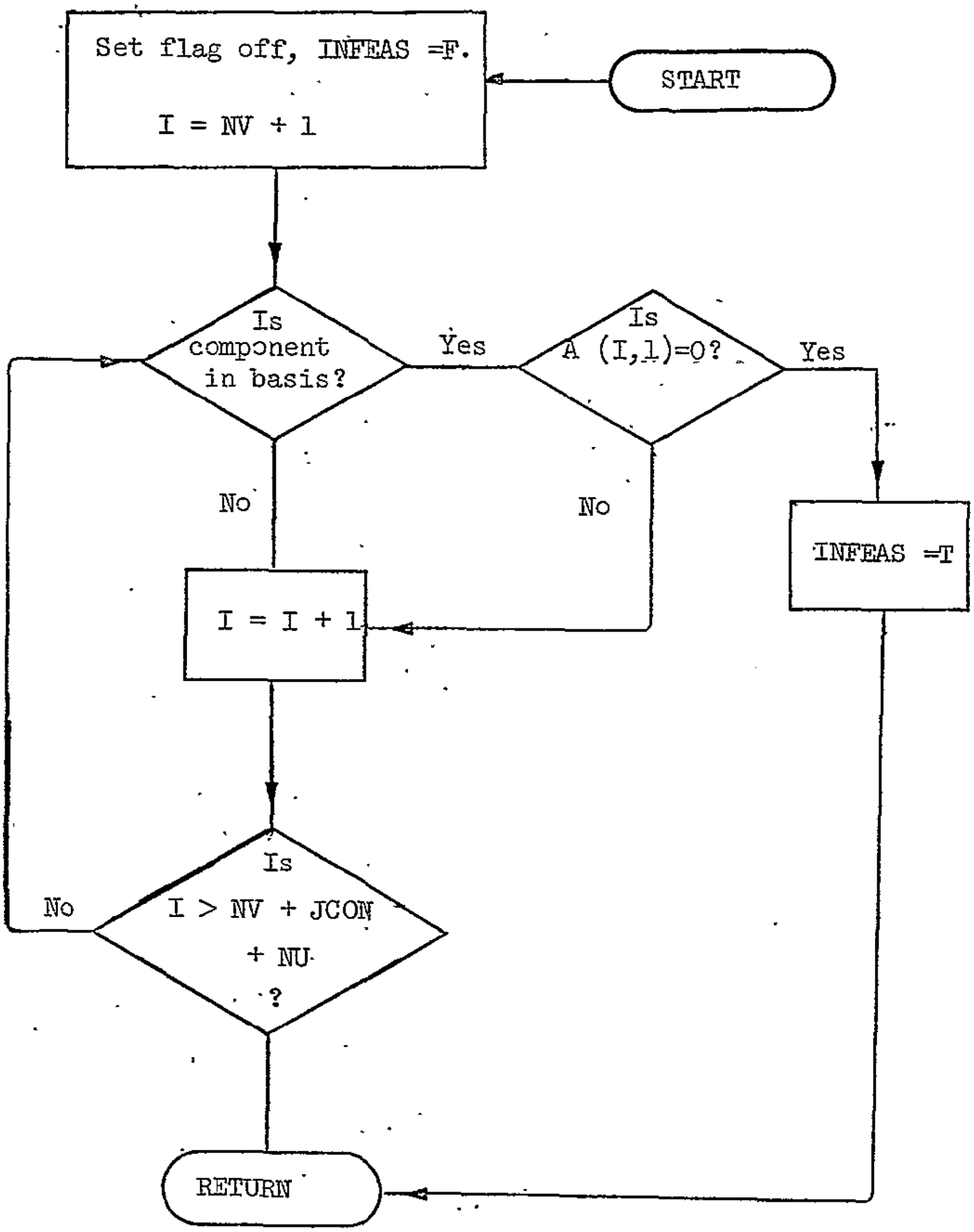

2
Read A matrix (first tableau)
Initialize pointers for basis component.
B
Find restricted component with maximum partial deviative SUBROUTINE FINMV
Determine whether transformation is to be Type I or Type II: SUB MAXIND
Find restricted variables to enter basis?
Yes
Write descriptive information
Go to D
No
C
Find unrestricted component to enter basis: SUBROUTINE FINMV
Is solution optimal?
Yes
Go to E
No
3


3
Find unrestricted component to enter basis?
No
Go to B
Yes
D
Make transformation: SUBROUTINE TRAN
Increment iteration counter
Was component entering basis restricted?
No
Yes
Write descriptive information
Check for degeneracy: SUBROUTINE DGCK
4

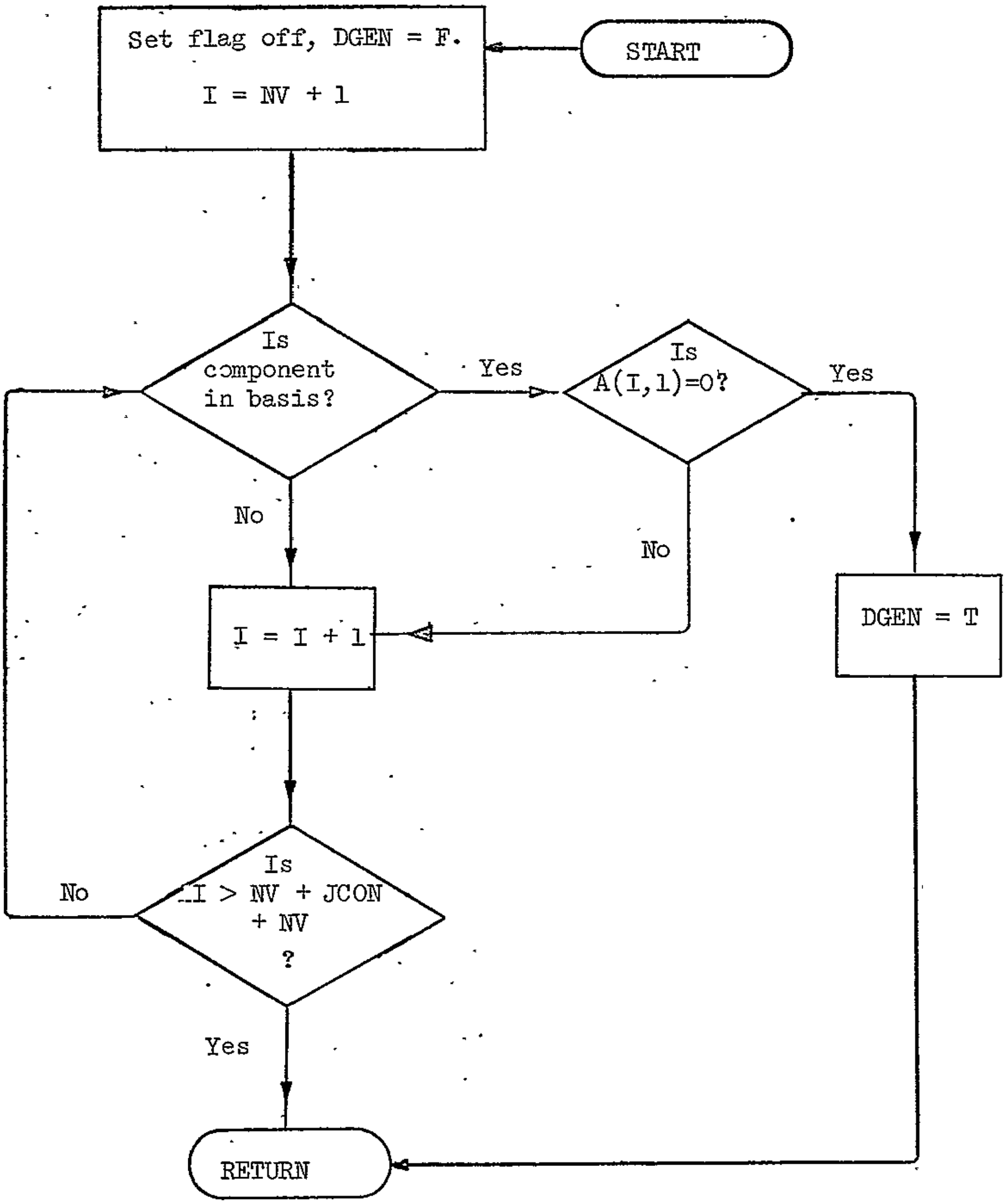

4
Is solution degenerate ?
Yes
No
Write Message
Go to F
Check for infeasible solution:
SUBROUTINE INFCK
Is solution infeasible ?
Yes
No
Write Message
Go to F
Print tableau?
No
Yes
Write transformed A matrix
Go to C


E
Write optional solution
Write final A matrix (tableau)
Write elapsed time and number of iterations
F
Any more problems?
Yes
Go to A
No
STOP

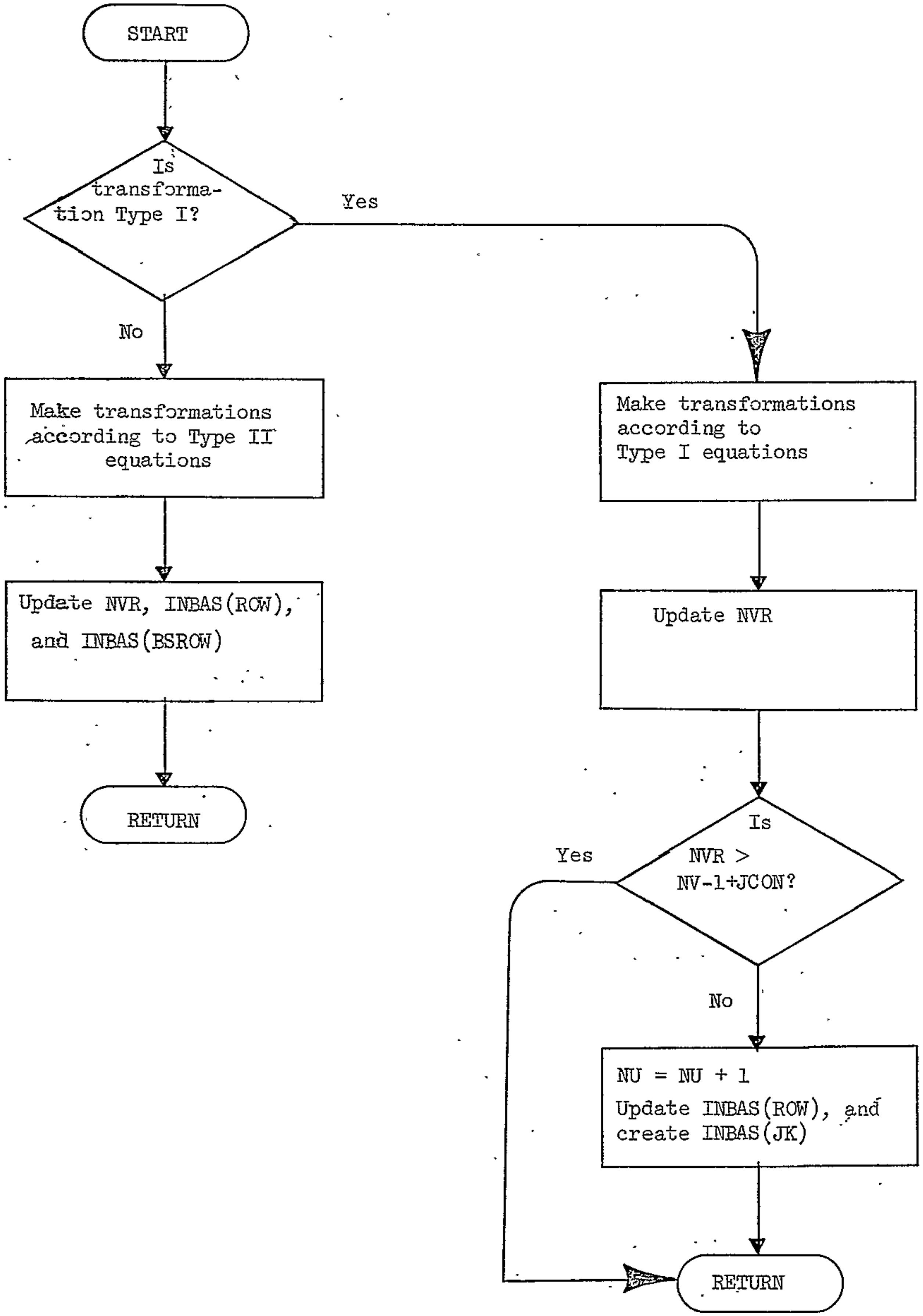
SUBROUTINE INFCK (INFEAS)



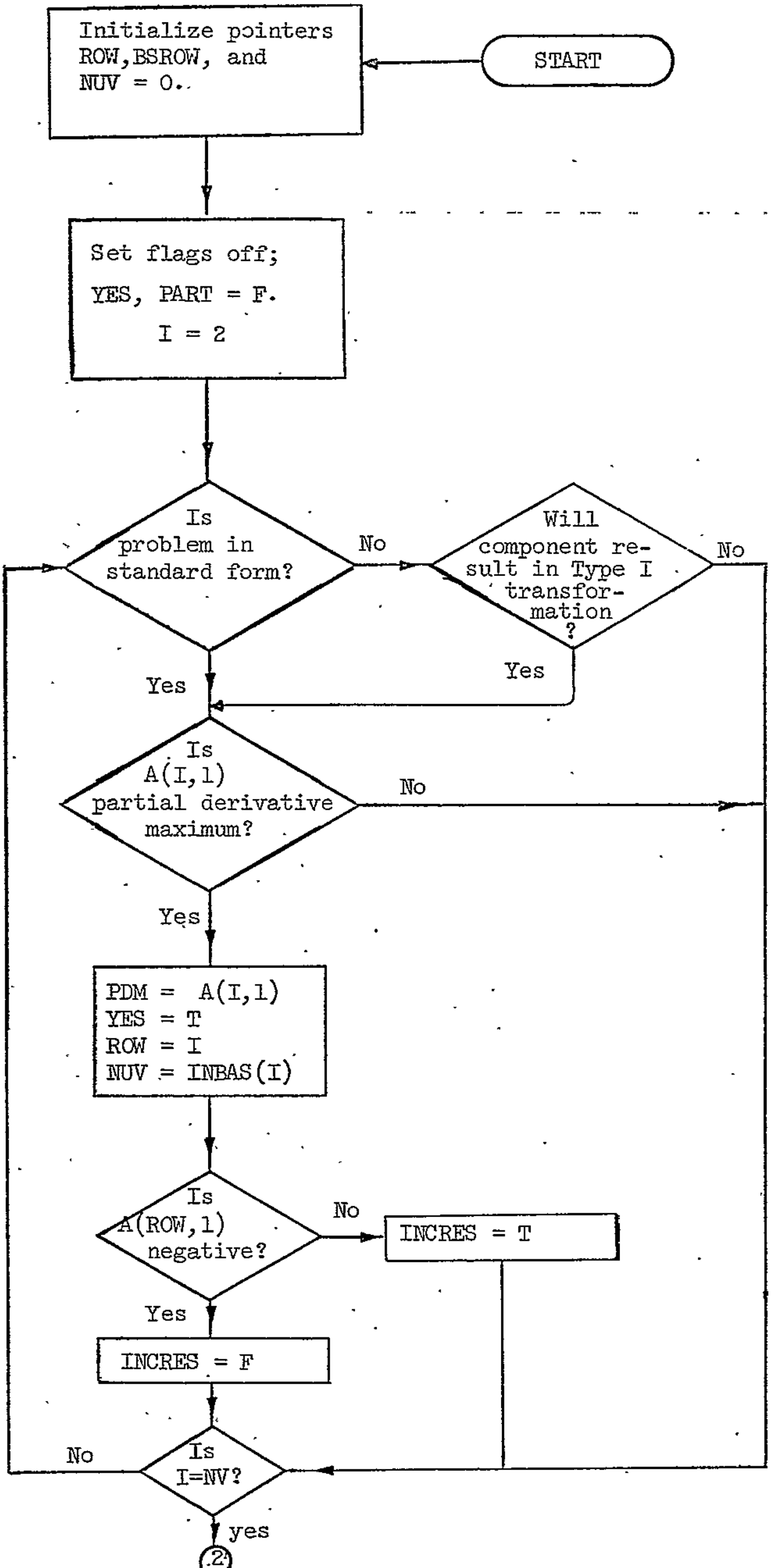
SUBROUTINE DGCK (DGEN)

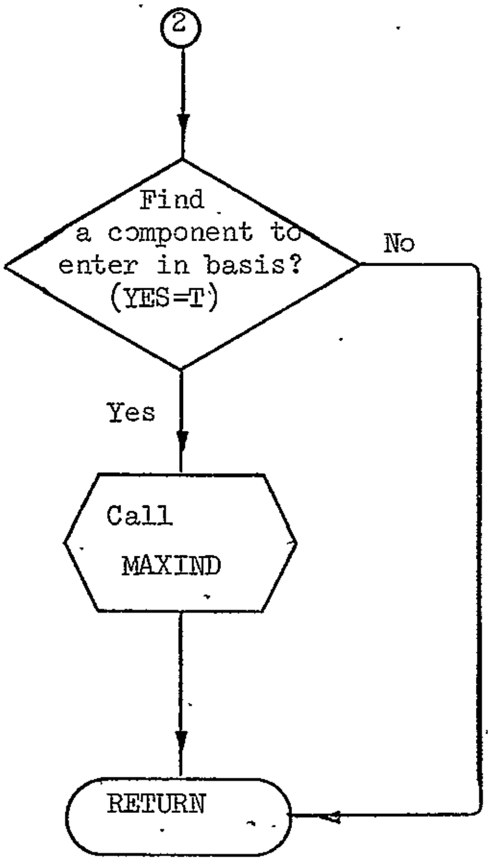


SUBROUTINE TRAN (ROW,BSROW,UV)

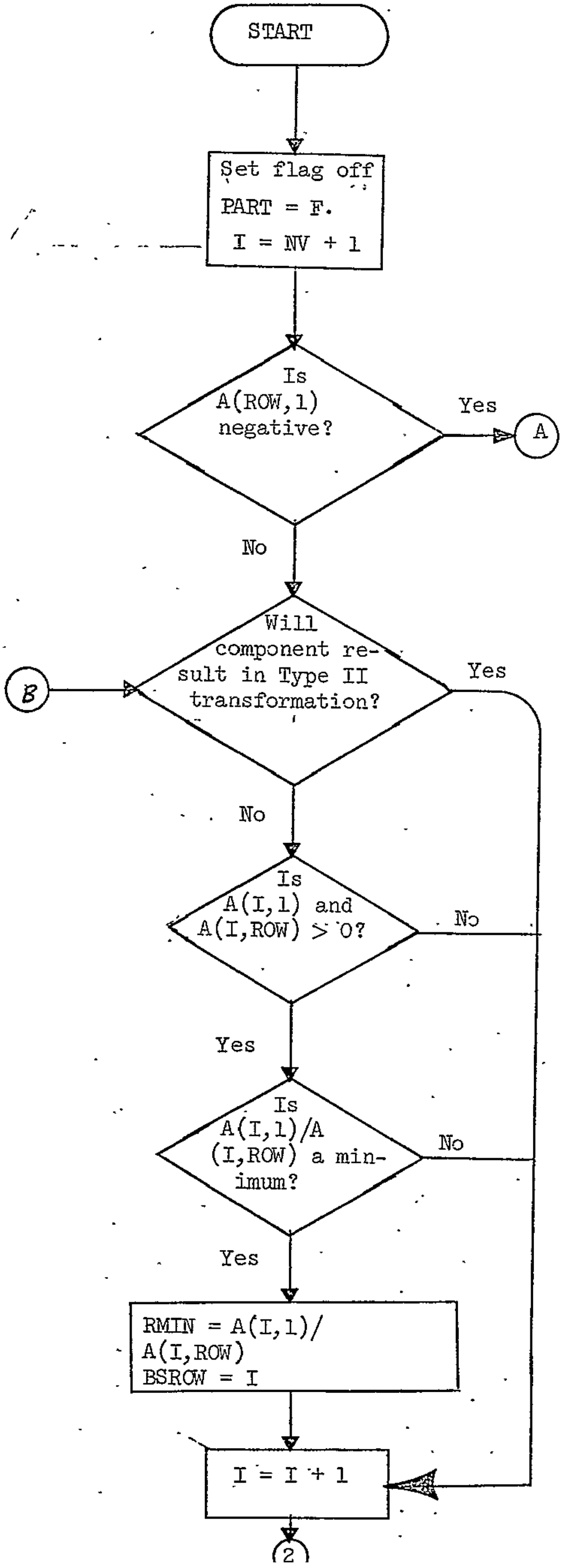

START
Is transforma-tion Type I?
Yes
No
Make transformations according to Type II equations
Update NVR, INBAS(ROW), and INBAS(BSROW)
RETURN
Make transformations according to Type I equations
Update NVR
Is NVR > NV-1+JCON?
Yes
No
NU = NU + 1
Update INBAS(ROW), and create INBAS(JK)
RETURN

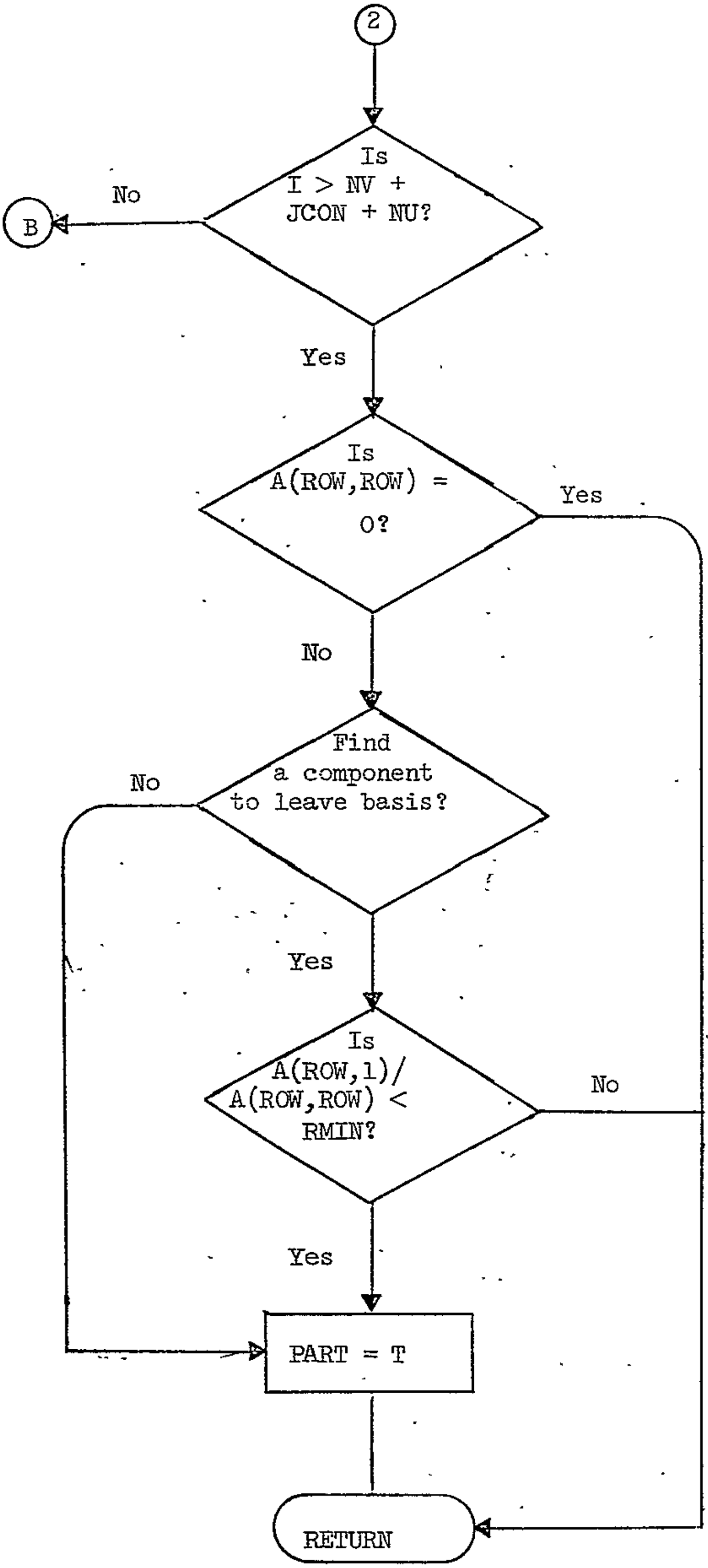
SUBROUTINE FINMV

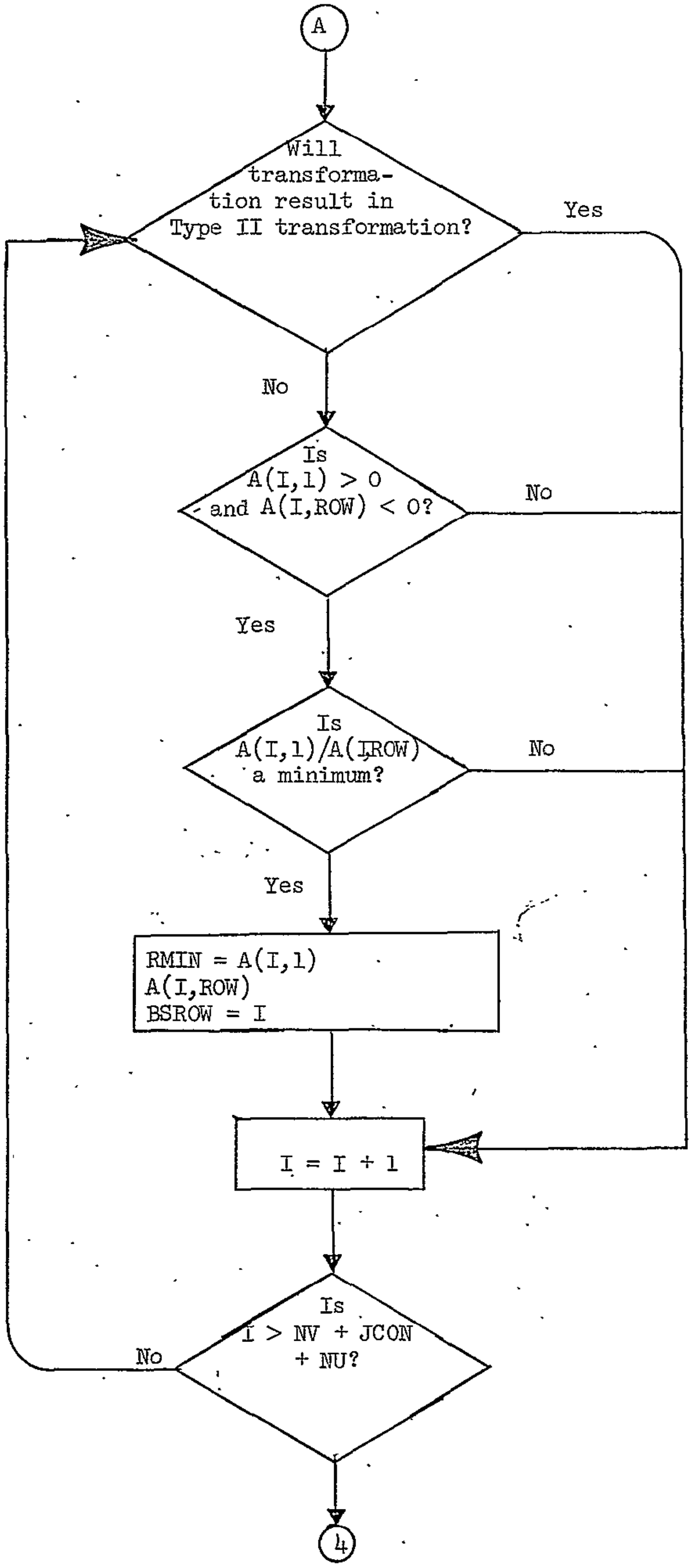

START
Initialize pointers ROW, BSROW, and NUV = 0.
Set flags off; YES, PART = F. I = 2
Is problem in standard form?
No
Will component result in Type I transformation ?
No
Yes
Yes
Is A(I,1) partial derivative maximum?
No
Yes
PDM = A(I,1)
YES = T
ROW = I
NUV = INBAS(I)
Is A(ROW,1) negative?
No
INCRES = T
Yes
INCRES = F
No
Is I=NV?
yes
2

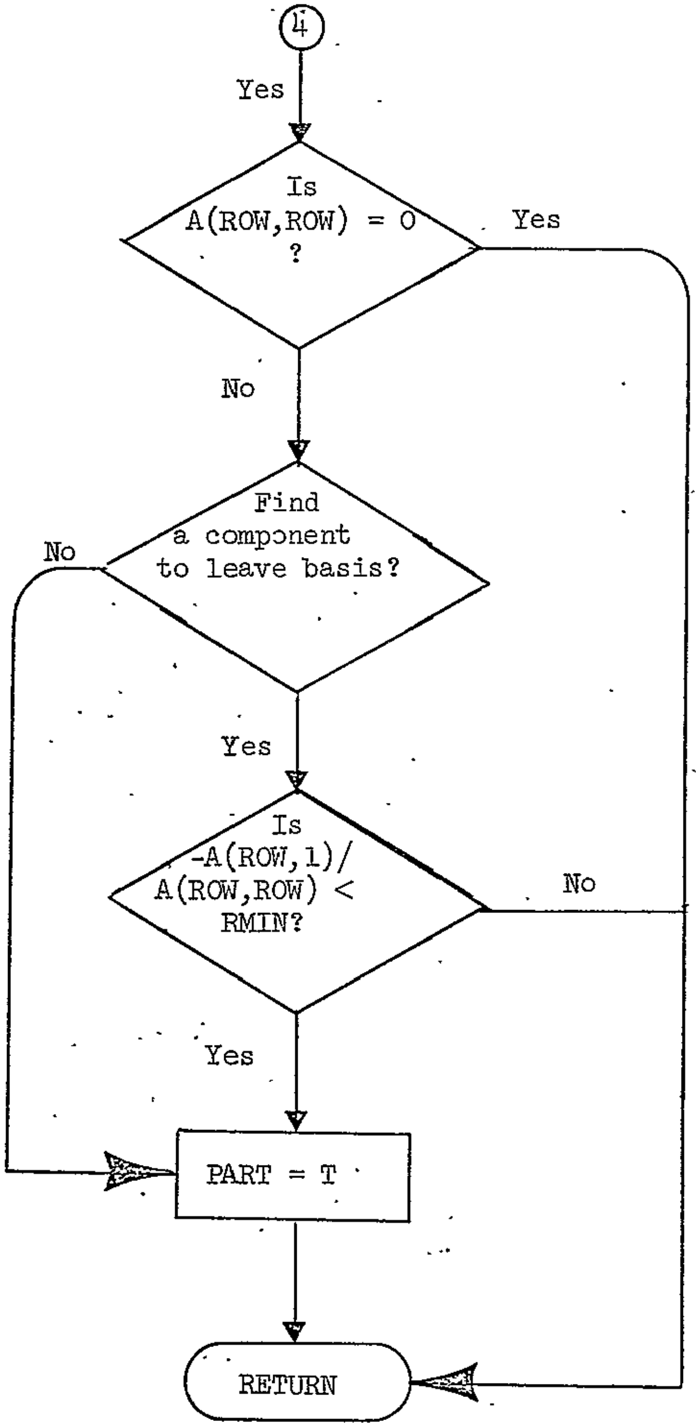

2
Find a component to enter in basis? (YES=T)
No
Yes
Call MAXIND
RETURN

SUBROUTINE MAXIND




2
Is I > NV + JCON + NU?
No
B
Yes
Is A(ROW,ROW) = 0?
Yes
No
Find a component to leave basis?
No
Yes
Is A(ROW,1)/ A(ROW,ROW) < RMIN?
No
Yes
PART = T
RETURN


A
Will transformation result in Type II transformation?
Yes
No
Is A(I,1) > 0 and A(I,ROW) < 0?
No
Yes
Is A(I,1)/A(I,ROW) a minimum?
No
Yes
RMIN = A(I,1)
A(I,ROW)
BSROW = I
I = I + 1
Is I > NV + JCON + NU?
No
4


4
Yes
Is A(ROW,ROW) = 0 ?
Yes
No
Find a component to leave basis?
No
Yes
Is -A(ROW,1)/ A(ROW,ROW) < RMIN?
No
Yes
PART = T
RETURN

BIBLIOGRAPHY

[1] Gordon Sherman and Stanley Reiter, "Discrete Optimizing", J. Soc. Indust. Appl. Math., Vol. 13, No. 3, September, 1965.

[2] J. A. Joseph, "Heuristic Approach to Nonstandard Form Assignment Problems", Operations Research, Vol. 15, No. 4, July-August 1967.

[3] J. T. Robacker, "Some Experiments on the Traveling Salesman Problem", RAND Research Report RM-1521, 1955.

[4] G. A. Croes, "A Method for Solving Traveling Salesman Problems", Operations Research, 6 (1958), pp. 790-812.

[5] M. Held and R. M. Karp, "A Dynamic Programming Approach to Sequencing Problems", J. Soc. Indust. Appl. Math., 10 (1962), pp. 196-210.

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

[7] R. L. Karge and G. L. Thompson, "A Heuristic Approach to Traveling Salesman Problems", Management Sci., 10(1964), pp. 225-248.

[8] Kubert, J. Szabo and S. Giulieri, "The Perspective Representation of Functions of Two Variables", Journal of the Association for Computing Machinery, Vol. 15, No. 2, April 1960, pp. 193-204.

[9] John McGraw Rooker, "An Experimental Study of Some Quadratic Programming Algorithms", Technical Report No. 7, University of Tennessee Computing Center, Knoxville, Tennessee.

[10] Isaac J. Schoenberg, "The Relaxation Method for Linear Inequalities", Canadian Journal of Mathematics, 1954, Vol. 6, pp. 393-404.

[11] A. Shimbel "Applications of Matrix Algebra to Communication Nets" Bulletin of Mathematical Biophysics, Vol. 13, 1951..

[12] Hillier and Leberman, Introduction to Operations Research, pp. 218-222.