AN ALGORITHM FOR COMPUTING THE ALPHA-WIDTH OF (0,1) MATRICES

WALTER L. STANLEY

AN ALGORITHM FOR COMPUTING THE ALPHA-WIDTH OF (0,1) MATRICES

* * * * *

Walter L. Stanley

AN ALGORITHM FOR COMPUTING THE ALPHA-WIDTH OF (0,1) MATRICES

by

Walter L. Stanley

Lieutenant, United States Navy

Submitted in partial fulfillment of

the requirements for the degree of

MASTER OF SCIENCE
IN
OPERATIONS RESEARCH

United States Naval Postgraduate School
Monterey, California

1 9 6 5

AN ALGORITHM FOR COMPUTING THE ALPHA-WIDTH OF (0,1) MATRICES

by

Walter L. Stanley

This work is accepted as fulfilling

the thesis requirements for the degree of

MASTER OF SCIENCE

IN

OPERATIONS RESEARCH

from the

United States Naval Postgraduate School

ABSTRACT

A branch and bound technique is used to derive an algorithm for computing the alpha-width of any matrix of zeros and ones. Through computation of the 1-width of over 200 matrices of various dimensions, it is found that less than 20 minutes of computation time on the Control Data 1604 digital computer is required to complete the computation for most matrices. Applications of the algorithm to integer programming and to various targeting problems are described. Extensions are suggested for computing the minimal cost alpha-width, and for computing a minimal C-cover.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

TABLE OF SYMBOLS

$x \in X$	x is a member of X
$\sum_{j=1}^{n} x_j$	$x_1 + x_2 + \dots + x_n$
$(a_1,a_2,\dots,a_n)^T$	The <u>transpose</u> of the matrix (or vector), $(a_1,a_2,\dots,a_n)$ $(a_1, \dots, a_n)^T = \begin{matrix} a_1 \\ a_2 \\ \cdot \\ \cdot \\ \cdot \\ a_n \end{matrix}$
$A \prec B$	A is <u>majorized</u> by B
$A \not\prec B$	A is <u>not majorized</u> by B
$\overline{A}$	The <u>maximal</u> matrix with row sums R
$[e_{ij}]$	The matrix, E
$\vec{0}$	The vector, (0,0,...,0)
$B \sim A$	The <u>relative complement</u> of A with respect to B
$A \cup B$	The <u>union</u> of A and B
$A \cap B$	The <u>intersection</u> of A and B
$X,\overline{X}$	A <u>partition</u> of a set Y satisfying: $X \cap \overline{X} = 0$, $X \cup \overline{X} = Y$
$\binom{n}{k}$	The binomial coefficient. $\binom{n}{k} = n!/(k!(n-k)!)$
p_1 & p_2	p_1 <u>and</u> p_2 (logical)
$p_1 + p_2$	p_1 <u>or</u> p_2 (logical) section 7.3 only
$\overline{p_1}$	<u>not</u> p_1 (logical)

1. A Targeting Problem.

Consider the following rather specialized targeting problem: a communications network is given (Fig. 1), in which stations can communicate directly only with those stations to which they are connected by a link. Of course, this would be the case with any kind of land line network, but it is possible also, in the case of UHF radio communications, micro-wave relay systems, and even signal light. We ask this question: what is the minimum number of stations that must be destroyed so that the network is totally disrupted; that is, so that no pair of surviving stations can communicate?

The answer is given in Figure 2; in which those stations targeted have been crossed out. It is perhaps surprising to note that the station most central to the network; the one directly connected to the greatest number of stations, is not targeted. In fact, if this station were included in the target list, we should be forced to target the four indicated targets anyway, and thus we would have been forced away from the optimal solution.

Let us note, parenthetically, that no claim is made that our targeting policy is the best one. It is quite probably válid, and indeed optimal, if the purpose of the attack is, for example, the total (and temporary) disruption of an enemy's warning system for the protection of a second strike to follow immediately. But assume that the network is a railroad system. It is quite possible that a policy of bombing junctions, switching yards, and accessible rail lines would have little lasting effect on the effectiveness of the transportation system because of the ready availability of repair

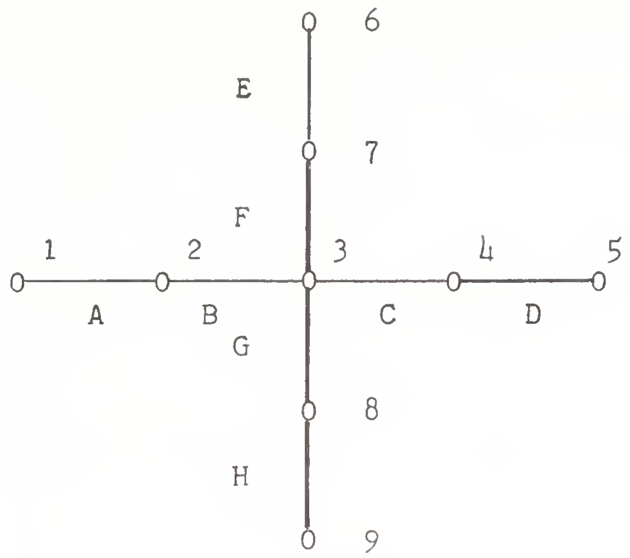


Figure 1

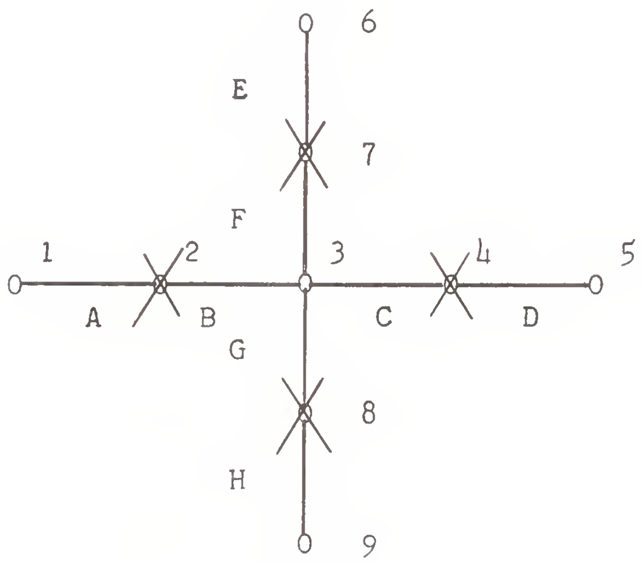


Figure 2

equipment and personnel. For example, in the interdiction of the French railroads prior to the invasion of Normandy in World War II; our bombing of marshalling yards and other junctions caused little disruption of rail traffic, although it did strain the repair capabilities of the rail system severely. On the other hand, when bridges over the Seine, Oise, and Meuse Rivers were added to the target list, results were spectacular. On 26 May, all routes over the Seine north of Paris were closed to rail traffic and remained closed for the next thirty days. By contrast, marshalling yards could be repaired in one or two days. (See pp 217-230; and especially p 228 of [6]).

No matter, this simple problem will serve to illustrate the very general algorithm to be described in section 3; without requiring that cumbersome set-up procedures be learned before getting down to work.

The solution to this targeting problem was obtained without difficulty after the initial error of trying to include the central station (number 3) in the target list, merely by inspection of the network layout. It is unfortunate that so few communications networks of nine stations and eight connecting links are of interest in a problem of this type. Clearly, if a network of interesting size were examined (let us say on the order of 15 stations and 35 connecting links), the solution by inspection would be quite difficult. Where, then, are we to look for a method of attack on this problem?

It is well known from the theory of graphs, that every graph may be represented by an incidence matrix of zeros and ones; in fact

by any of several incidence matrices depending upon the purpose for the representation. [1]. For the purpose of this paper we will use the following terminology from graph theory: a node of a graph is the junction of two or more links of the graph (synonym: vertex); an arc is a link between two nodes, and in this paper will be considered to be without direction. We define the node-arc incidence matrix, A, of a graph, by construction as follows: List the nodes of the graph horizontally and the arcs vertically so that they are labels of columns and rows of the matrix, respectively. If the j^{th} node is a terminal point of the i^{th} arc, set $a_{ij} = 1$. Otherwise, set $a_{ij} = 0$. The node-arc incidence matrix of the communications network of Figure 1 is displayed in Figure 3.

The targeting problem restated in graph theoretic terms is: Find the minimum number of nodes so that each arc of the graph has at least one of the nodes as a terminal. Since we already know the answer to this simple problem, it would be well to examine this solution applied to the node-arc incidence matrix. We construct a new matrix from the incidence matrix by including only those columns labelled with one of the nodes in the solution set. This matrix is displayed in Figure 4. It contains the same number of rows as the original matrix, but has only four columns. We note that whereas there were two "1"'s in each of the rows of the incidence matrix (one for each terminal of each arc); there is only one "1" in the sub-matrix.

A little reflection upon the above observation leads to a third formulation of the targeting problem: given the node-arc incidence matrix of the communications network, find the smallest subset of

	1	2	3	4	5	6	7	8	9
A	1	1	0	0	0	0	0	0	0
B	0	1	1	0	0	0	0	0	0
C	0	0	1	1	0	0	0	0	0
D	0	0	0	1	1	0	0	0	0
E	0	0	0	0	0	1	1	0	0
F	0	0	1	0	0	0	1	0	0
G	0	0	1	0	0	0	0	1	0
H	0	0	0	0	0	0	0	1	1

Figure 3

	2	4	7	8
A	1	0	0	0
B	1	0	0	0
C	0	1	0	0
D	0	1	0	0
E	0	0	1	0
F	0	0	1	0
G	0	0	0	1
H	0	0	0	1

Figure 4

columns of the matrix with the property that each row is represented by at least one "1" in this subset of columns. But this smallest subset of columns is precisely what Fulkerson and Ryser call a minimal set of representatives for the (0,1) matrix, A; and the cardinality of this set is called the "width" of A. [4]. The problem may be generalized: we require that each row of the matrix be represented by at least alpha "1"'s (where alpha is a positive integer). We shall use the terminology "minimal α-set of representatives for the (0,1) matrix, A"; and "α-width of A". This terminology is due also to Fulkerson and Ryser. [4].

Thus the simple targeting problem may be solved by finding the 1-width of the node-arc incidence matrix of the communications network. It is the purpose of this paper to present an algorithm for finding the α-width of any (0,1) matrix; and for specifying at least one minimal α-set of representatives for that matrix. Since we already have solved one problem of this type, we shall use this communications network and its associated incidence matrix for illustrative purposes throughout the balance of this paper.

We now state the general problem which we desire to solve: given a finite set, X, and a class, Y, of k non-empty subsets of X (but not necessarily the class of all non-empty subsets of X), find a sub-class, Z, of Y, with the property that if $x \in X$, then x is a member of at least α of the members of Z. This is a quite general problem, as will be shown in later sections of this paper. Any problem which can be formulated in the terms specified in this paragraph is capable of being solved by the algorithm to be presented. The incidence matrix for this abstract problem is constructed by listing members of

X vertically and subsets of X horizontally. Then we place a "1" in the i^{th} row and j^{th} column if the i^{th} member of X is a member of the j^{th} subset of X.

2. The Class, $\mathfrak{U}(R,S)$.

Let A denote the (0,1) matrix of size m by n; that is, A is a matrix with m rows and n columns, each of whose elements is either zero or one. Let the sum of all of the elements of the i^{th} row be denoted by r_i; and the sum of all of the elements in the j^{th} column be denoted by s_j. That is:

$$(1) \qquad \sum_{j=1}^{n} a_{ij} = r_i \qquad (i=1, \ldots, m)$$

$$(2) \qquad \sum_{i=1}^{m} a_{ij} = s_j \qquad (j=1, 2, \ldots, n)$$

We note that $\sum_{i=1}^{m} r_i = \sum_{j=1}^{n} s_j$.

We call the column vector, $(r_1, r_2, \ldots, r_m)^T = R$, the row sum vector; and the row vector, $(s_1, s_2, \ldots, s_n) = S$, the column sum vector. We denote by $\mathfrak{U}(R,S)$ the class of all (0,1) matrices of size m by n with row sum vector, and column sum vector, R and S, respectively.

From the class, $\mathfrak{U}(R,S)$, many very interesting combinatorial results may be obtained. An excellent survey of this material may be found in Ryser.[10]. We will be concerned primarily with a parameter, $\tilde{\varepsilon}$, or $\tilde{\varepsilon}(\alpha)$, of the class, which is defined as the greatest lower bound on the α-width of any matrix in $\mathfrak{U}(R,S)$. That is, $\tilde{\varepsilon}$ is the α-width of the matrix in $\mathfrak{U}(R,S)$ which has the smallest α-width of any matrix in the class.

Although not of concern until a later section, it will be of interest to determine under what conditions the class, $\mathfrak{U}(R,S)$ is non-empty. Let $\delta_i = (1, 1, \ldots, 1, 1, 0, 0, \ldots, 0)$ be an n-dimensional vector with the first r_i components equal to one, and the remaining $n - r_i$ components equal to zero. We then define a matrix of the form,

$$\bar{A} = \begin{bmatrix} \delta_1 \\ \delta_2 \\ \cdot \\ \cdot \\ \delta_m \end{bmatrix}$$

called the maximal matrix with row sum vector, R. It has column sum vector, $\bar{S} = (\bar{s}_1, \bar{s}_2, \ldots, \bar{s}_n)$. Now since $\sum_{i=1}^{m} r_i = \sum_{j=1}^{n} \bar{s}_j$; for R fixed, $\bar{S}$ is unique, by definition of the δ_i, and the class $\mathfrak{U}(R,\bar{S})$, by a simple contradiction argument, has only one member; namely, $\bar{A}$.

Let $Q = (q_1, q_2, \ldots, q_k)$ and $Q^* = (q_1^*, q_2^*, \ldots, q_k^*)$ be any two k-dimensional vectors whose components are non-negative integers. We say that Q is <u>majorized</u> by Q*, denoted $Q \prec Q^*$, provided that with subscripts renumbered so that $q_1 \geqq q_2 \geqq \cdots \geqq q_k$; and $q_1^* \geqq q_2^* \geqq \cdots \geqq q_k^*$, the following statements are true:

(3) $\quad q_1 + q_2 + \cdots + q_j \leqq q_1^* + q_2^* + \cdots + q_j^* \quad (j=1,2,\ldots,k-1)$

(4) $\quad q_1 + q_2 + \cdots + q_k = q_1^* + q_2^* + \cdots + q_k^*$

We say that Q is <u>normalized</u> if $q_1 \geqq q_2 \geqq \cdots \geqq q_k$. These two definitions now enable us to give conditions under which $\mathfrak{U}(R,S)$ is non-empty.

<u>Theorem 2.1</u>

Let $R = (r_1, r_2, \ldots, r_m)$, and $S = (s_1, s_2, \ldots, s_n)$ be two normalized vectors whose components are non-negative integers, and such that $\sum_{i=1}^{m} r_i = \sum_{j=1}^{n} s_j$. Let $\bar{A}$ be the maximal matrix of size m by n, with row sum vector, R, and column sum vector $\bar{S} = (\bar{s}_1, \bar{s}_2, \ldots, \bar{s}_n)$.

Then a necessary and sufficient condition that $\mathfrak{U}(R,S)$ be non-empty is that $S \prec \bar{S}$.

Proof: Assume $S \not\prec \bar{S}$. Since $\sum_{j=1}^{n} s_j = \sum_{j=1}^{n} \bar{s}_j = \sum_{i=1}^{m} r_i$, it must be that $S \not\prec S$ because equation (3), above, is violated, that is, for some k, it must be the case that $s_1 + s_2 + \ldots + s_k > \bar{s}_1 + \bar{s}_2 + \ldots + \bar{s}_k$. But then $\bar{A}$ is not maximal, since the first k columns of A contain more "1"'s than the first k columns of $\bar{A}$. The hypothesis is that A is maximal, so we have arrived at a contradiction, thus demonstrating the necessity of the theorem.

To show sufficiency; we shall construct a matrix, $A \in \mathfrak{U}(R,S)$ from the maxtrix $\bar{A}$. This construction is due to Ryser. [9] . The construction will proceed by shifting ones in the i^{th} row of $\bar{A}$ to other positions in the same row. We note again, that R, S, and $\bar{S}$ are all normalized, and that $S \prec \bar{S}$. If $s_1 < \bar{s}_1$, rearrange the ones in the rows of A so that only s_1 ones remain in the first column. We may do this unless $s_j > s_1$ (j=2, ..., n), in which case, $\bar{s}_1 + \bar{s}_2 + \ldots + \bar{s}_n > n \cdot s_1 \geqq s_1 + s_2 + \ldots + s_n = \bar{s}_1 + \ldots + \bar{s}_n$; an absurdity. We continue by induction. Suppose that the first t columns of A have been rearranged. The matrix thus far constructed has the form,

$$A' = \left[b_1 \; b_2 \; \ldots \; b_t \; b_{t+1} \; \ldots \; b_n \right]$$

where there are s_j ones in the j^{th} column of A' (j = 1, ..., t).

We now construct the $(t+1)^{st}$ column. Let the number of ones in the j^{th} column be s_j' (j = t+1, ..., n). We may construct A' without loss of generality such that, $s_{t+1}' \geqq \ldots \geqq s_n$. Now it is possible that either $s_{t+1} < s_{t+1}'$, or that $s_{t+1} > s_{t+1}'$. We consider each case in turn:

Case I: $s_{t+1} < s'_{t+1}$

Remove ones from the $(t+1)^{st}$ column, placing them in other columns to the right. If sufficiently many ones may be removed by this procedure, the column of A is constructed, and we are finished. Suppose therefore, that there remain, d ones in column t+1, so that $s_{t+1} < d \leqq s'_{t+1}$. Let the matrix at this stage be denoted by $[e_{rs}]$. Now if $d > s_{t+1}$, then for every $e_{r,t+1} = 1$; we must have $e_{rj} = 1$ $(j = t+2, \ldots, n)$. Hence $s_{t+1} + \ldots + s_n$ must at least equal $d \cdot (n-t)$. But $s_{t+1} < d$; $s_{t+2} \leqq s_{t+1} < d$; etc., so that

$$s_{t+1} + \ldots + s_n < (n-t) \cdot d \leqq s_{t+1} + \ldots + s_n$$

an absurdity.

Case II: $s_{t+1} > s'_{t+1}$.

Insert ones in the $(t+1)^{st}$ column from columns to the right. If sufficiently many ones can be inserted, we are finished. We therefore assume that sufficiently many ones cannot be inserted by this procedure; in fact, we assume that column t+1 contains only d ones such that $s'_{t+1} \leqq d < s_{t+1}$. Again, let the matrix at this stage of construction be denoted by $[e_{rs}]$. Then if $e_{r,t+1} = 0$, it must be the case that $e_{rj} = 0$ $(j = t+1, \ldots, n)$. Now suppose that $e_{qj} = 1$ for some $j \geqq t+2$. Then either $e_{qk} = 1$ for all $k \leqq t+1$, or else, for some $k \leqq t$, $e_{qk} = 0$. Consider the case in which $e_{qk} = 0$. Since $s_k \geqq s_{t+1} > d$, there must exist $e_{pk} = 1$, and $e_{q,t+1} = 0$. We interchange e_{qj} and e_{qk}; and also interchange e_{pk} and $e_{p,t+1}$. This increases the value of d by one without changing the value of any column sum for columns to the left of column t+1. Suppose we make all such interchanges and still, $d < s_{t+1}$. This situation includes the case mentioned above, that $e_{qk} = 1$ for all $k \leqq t+1$. It is no longer possible to shift ones from columns t+2, ..., n; into columns 1, ..., t+1. This must mean that

either all of the ones for a given row are in columns to the left of column t+2; or that all of elements of a given row to the left of column t+2 are equal to one. In either case, it must be that,

$$s_1 + \dots + s_t + \underline{d} = s_1 + \dots + s_t + s_{t+1}$$

But then, since $S \prec \bar{S}$,

$$s_1 + \dots + s_{t+1} \leqq \bar{s}_1 + \dots + \bar{s}_{t+1} \quad (= s_1 + \dots + s_t + d)$$

whence $s_{t+1} \leqq d$; contrary to the assumption. QED

We now consider an extension of the concept of α-width. Let $C = (c_1, \dots, c_m)$ be an m-dimensional vector of non-negative integers. We wish to find the smallest subset of columns of $A \in \mathcal{U}(R,S)$ such that the i^{th} row of A is represented by at least c_i ones in this subset of columns. Such a subset of columns will be called a minimal C-cover for A, and we shall denote its cardinality by $\varepsilon(C)$, called the C-width of A. Clearly, if $C = (\alpha, \dots, \alpha)$, then $\varepsilon(C) = \varepsilon(\alpha)$. We define $\tilde{\varepsilon}(C)$ to be the greatest lower bound on the C-width of any matrix in $\mathcal{U}(R,S)$. $\tilde{\varepsilon}(C)$ can be estimated by $\rho(C)$ as follows:

(5) $\rho(C)$ = the smallest integer such that $\sum_{j=1}^{\rho} s_j \geqq \sum_{i=1}^{m} c_i$.

We shall use this formula in the algorithm to be presented in section three. Note that if $C = (\alpha, \dots, \alpha)$; then $\rho(C)$ = the smallest integer such that $\sum_{j=1}^{\rho} s_j \geqq m \cdot \alpha$

3. Derivation of the Algorithm.

We shall now describe our algorithm for finding the α-width of a (0,1) matrix. The branch and bound technique was suggested to me by D. R. Fulkerson of the RAND Corporation, Santa Monica, California, and is patterned after the branch and bound solution to the travelling salesman problem designed by Little, et alii. [7]

We have several techniques for estimating $\varepsilon(\alpha)$(which we shall henceforth call the α-width of the class, $\mathcal{U}(R,S)$). One such technique is described in the preceding section, in which we compute the parameter, ρ. Now a given (0,1) matrix of size m by n, is a member of a class, $\mathcal{U}(R,S)$. We can partition the class, $\mathcal{U}(R,S)$ into two sub-classes, one consisting of those matrices which have a selected column, say column p, as a member of a minimal α-set; and the other consisting of those matrices for which column p is not a member of any minimal α-set for the matrix.

We thus have two sub-classes, each of which has no more members than the original class, and we know that the original matrix must be in one, and only one of the sub-classes. Consider the sub-class whose matrices have column p as a member of a minimal α-set. We may use this information to reduce the dimensions of all the matrices in the class as follows: if $\alpha = 1$, then every row which has a one in column p is adequately represented by column p, and needs not be considered subsequently. If $\alpha \neq 1$, we still may note that these same rows are represented once by column p, and thus need be represented only $\alpha - 1$ more times subsequently. Furthermore, we have made a "decision" about column p, namely that it is included in a minimal α-set of all matrices in this sub-class. We may thus reduce the dimensions of all matrices of the sub-class by one column, and (if $\alpha = 1$) by a number of rows. If $\alpha \neq 1$, we will keep track of those rows which yet need only $\alpha - 1$ representatives. Hence we have for this class a vector, C, whose components are either α, or $\alpha - 1$.

We may also reduce the dimensions of the matrices in the other sub-class by one column, for we have made a "decision" for this sub-class

namely, that no α-set contains column p, for any matrix of the sub-class. Hence every row of the matrices of this sub-class needs to be subsequently represented α times, regardless of the value of α.

Now let us estimate $\tilde{\varepsilon}$ for each of the two sub-classes. It is clear that these two numbers are both estimates of the α-width of the original matrix. We may actually improve the estimate for the first sub-class discussed, by adding one to the estimate of $\tilde{\varepsilon}$ for the sub-class. This is to account for the inclusion of column p in any α-set for any matrix in this sub-class. Now, it is certain that the smaller of these two numbers is not greater than the α-width of the original matrix.

Let us examine the sub-class corresponding to the smaller of the two estimates. We may partition this sub-class into two sub-classes, and so forth, until finally, some sub-class will be so small as to contain a unique matrix whose C-width we can determine by inspection. Part of such a continuing procedure is represented by the tree structure of Figure 5.

Now at any point in the procedure, the set of junctions (Fig. 5) which have no lines leading toward another junction represent a partition of the class to which the original matrix belongs into two or more sub-classes. By an obvious extension of the above discussion, the smallest of the several estimates for the α-width of the original matrix is not larger than the α-width of the original matrix. We may then focus our attention on the sub-class corresponding to this smallest estimate, branching out from the corresponding junction until one of the earlier estimates of ε is smaller than any of the most recently constructed estimates. Now let us set up a formal algorithm based upon the preceding discussion.

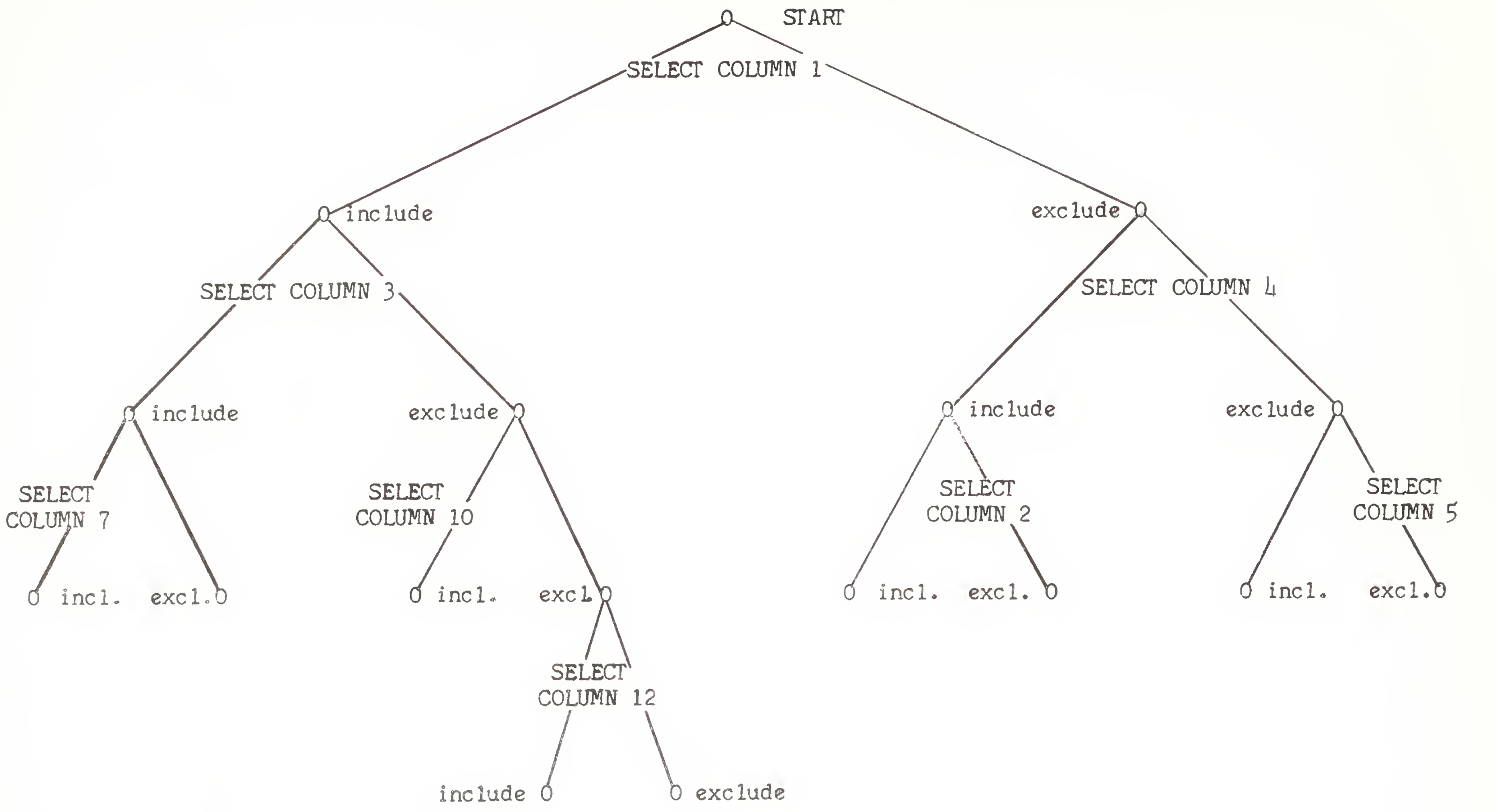

START
SELECT COLUMN 1
include
exclude
SELECT COLUMN 3
SELECT COLUMN 4
include
exclude
include
exclude
SELECT COLUMN 7
SELECT COLUMN 10
SELECT COLUMN 2
SELECT COLUMN 5
incl.
excl.
incl.
excl.
incl.
excl.
incl.
excl.
SELECT COLUMN 12
include
exclude

Figure 5

Let the (0,1) matrix, $A \in \mathfrak{A}(R,S)$, be given, with dimensions m by n. A matrix is said to be normalized when both its row sum and column sum vectors are normalized, and when the elements of the matrix have been rearranged so as to fit the new row and column sum vectors. Clearly, we lose no generality by considering only normalized matrices. Therefore, throughout the remainder of this paper, we assume that all matrices and sub-matrices have been normalized as part of the operation of constructing them.

Notation will, of necessity, become rather cumbersome, and for that reason, we now present such notation as we shall need in this section. There will be certain preliminary steps which serve to decrease the amount of work required in the main part of the algorithm, and since these preliminary steps are not always applicable, we shall assume that the given matrix, $A \in \mathfrak{A}(R,S)$, is the one with which we shall enter the main part of the algorithm.

The procedure in the algorithm is basically broken into two parts; (1) selecting a column for inspection and deriving the two sub-classes corresponding to the inclusion in, and the exclusion from the α-set of the selected column (the "branch" portion); and (2) estimating ε from each sub-class and choosing among all estimates, the smallest for the next iteration (the "bound" portion). We shall carry out the "bound" portion of the procedure by calculating ρ for each of the sub-classes and adding to ρ, the number of columns previously included in the α-set on the current branch. Eventually we shall obtain a sub-class of matrices, one of whose dimensions is zero, and is thus, empty. Clearly, $\tilde{\varepsilon}$ for this sub-class is zero. We can make a test for completion at this point. If the test fails, we

continue the algorithm along some other branch. It can be seen that we shall derive an even number of different sub-matrices of A before we reach termination. We shall subscript these sub-matrices in the order in which they are derived. Associated with each of the sub-matrices, of course, will be a row sum vector, a column sum vector, an estimate of the C-width of the class to which the sub-matrix belongs, and an estimate of the α-width of A based upon the condition that it can be obtained by continuing along the branch from which we derived the present sub-matrix. Note that since we may discontinue consideration of one branch at any time, and return to a previously discontinued branch; the subscripts of the matrices which we shall derive at any point of the procedure bear no relation to the subscript of the matrix from which the derivation follows. This point will be made again during our step by step description of the algorithm.

Now, we subscript every parameter associated with a particular sub-matrix with the same sub-script as its associated sub-matrix. We shall also require a "label" for each sub-matrix, and the typical label will be of the form "$a,\bar{b},c,\bar{d},\ldots$". This label gives us the information that for each particular sub-matrix, every column of A which is present in the label has been branched upon; and those columns which appear unbarred are assumed to be included in the α-set, whereas those which appear with a bar over them are assumed to be excluded from the α-set. Thus, the a,b,c,d in the example label above represent positive integers which are the column numbers of the original A matrix. One further convention to which we shall adhere; an even sub-script is taken to mean that the latest column upon which we branched is considered to be included in the α-set associated with the sub-matrix,

and thus this column number will appear unbarred in the associated label. On the other hand, an odd subscript is taken to mean that the latest column branched upon is considered to be excluded from the α-set associated with the sub-matrix, and thus this column number will appear in the associated label with a bar over it. The described notation is summarized below:

$A_p \in \mathcal{U}(R_p, S_p)$ has dimensions m_p by n_p.

$R_p = (r_{p1}, r_{p2}, \ldots, r_{pm_p})^T$.

$S_p = (s_{p1}, s_{p2}, \ldots, s_{pn_p})$.

ρ_p = the minimum k such that $\sum_{j=1}^{k} s_{pj} \geqq \sum_{i=1}^{m} c_{pi}$

$\varepsilon'_p = \rho_p$ + the number of columns unbarred in the label of A_p.

It is obvious that much information must be recorded for each of several matrices. Although a structure similar to that of Figure 5 could be used, we suggest the format of Figure 6. This figure shows a typical matrix A_p and all of the required information associated with this matrix. It will be convenient to suppress zero elements of the matrix. Note that we list the subscripts of the columns of the original matrix along the top, and directly below that, the order of subscripts for the derived matrix, A_p. The order of subscripts for the rows of A_p is listed along the left side of the matrix, and R_p and S_p are listed along the right side and the bottom, respectively. At some convenient point we list ρ_p, ε'_p, and the label associated with the matrix.

In section four, we solve the targeting problem of section one using this algorithm. The reader may desire to read section four

concurrently with the description of the algorithm which follows.

3.1 Preliminary Steps.

P1. If $r_m < \alpha$; the α-width does not exist. We terminate, or else decide to look for an α-width in which α is a smaller integer than that which the original problem specified.

P2. If $r_m > \alpha$; go directly to step S1, in the main part of the algorithm.

	3	10	1	2	5	(column subscripts of A)	
	1	2	3	4	5	R_p	
1	1		1	1	1	4	
2	1	1	1			3	
3	1	1		1		3	
4		1			1	2	$\rho_p = 2$
5			1			1	
6				1		1	$\varepsilon_p' = 5$
S_p	3	3	3	3	2		"$\bar{4}$,7,6,8,$\bar{9}$"

Figure 6

P3. If $r_m = r_{m-1} = \ldots = r_{m-k} = \alpha$, for some k, $(0 \leq k \leq m-1)$; then it is evident that each "1" in any of these k+1 rows must belong to a column in the minimal α-set of representatives for A. Therefore, in each such row, say the i^{th}, for each j such that $a_{ij} = 1$, record that the j^{th} column is in the minimal α-set and delete the j^{th} column from the matrix. Let $C = (\alpha, \ldots, \alpha)$ be an m-dimensional vector. For each k such that $a_{kj} = 1$ subtract one from the k^{th} component of C.

When this has been done for all columns, j, deleted from the matrix, delete any row, i, for which $c_i \leq 0$. Finally, recompute new row sum and column sum vectors, normalize the new matrix, and proceed to step S1 in the main portion of the algorithm. Now the set of columns that has been deleted in this preliminary step will not again be explicitly mentioned. The reader is cautioned to remember to add these columns to the α-set computed in the next section in order to arrive at the true α-width of the matrix, A.

3.2 The Branch and Bound Algorithm.

S1. We are given $A \in \mathfrak{U}(R,S)$ which has been normalized. If C was not computed in step P3, let $C = (\alpha, \ldots, \alpha)$. Cross out column one of A. We shall branch on this column because it is the column with the largest column sum. This is an entirely arbitrary decision. We could branch on any column whatsoever, but it seems reasonable that the one with the largest column sum would be likely to be included in the α-set. A counterexample is easy to construct. In any case, it is now necessary to decide whether or not to include this column in the α-set.

S2. Let us denote the matrix, A by $[\delta_1\ \delta_2\ \ldots\ \delta_n]$. Construct $A_1 (= [\delta_2\ \delta_3\ \ldots\ \delta_n])$ and label it "1". We examine the consequences of excluding column one from the α-set. A_1 is of size m_1 by n_1 $(=n-1)$, and $A_1 \in \mathfrak{U}(R_1,S_1)$. Our decision means that we still must locate c_i representatives for each row, but that we may not use any of the "1"'s in the excluded column of A. Calculate ρ_1 by equation (2-5), and since no columns are unbarred in the label, let $\varepsilon_1' = \rho_1$.

S3. We next examine the consequences of including column one in the α-set. Construct A_2 by deleting column one from A, forming a

temporary R_2 vector, and deleting every row for which the component-wise difference of R and R_2 is greater than or equal to the corresponding component of C. That is, if $r_i - r_{2i} \geqq c_i$; delete row i. Clearly this can happen at this step only if $c_i = 1$. We have now reduced A by one column and perhaps some number of rows. This reduced matrix, when normalized is called A_2, and we now form the permanent vectors, R_2 and S_2. We label this sub-matrix, "1". ρ_2 is calculated by equation (2-5), and since column one is unbarred in the associated label, $\varepsilon'_2 = \rho_2 + 1$.

S4. We must now decide along which branch it will be most profitable to continue. We make the decision by choosing the sub-matrix associated with min $[\varepsilon'_1, \varepsilon'_2]$. If $\varepsilon'_1 = \varepsilon'_2$, the choice is arbitrary. When using the algorithm for hand computation, the best choice is probably that matrix with the greatest number of unbarred columns in the associated label, that is, in this case, A_2. Having made this decision, we set min $[\varepsilon'_1, \varepsilon'_2] = \infty$, so that the same branch will not be chosen again at a later stage. We proceed to step S5. All succeeding steps in the algorithm will be described in general terms.

S5. In the preceeding step, we decided to proceed using matrix A_L, say, with associated label, "$p,q,\bar{r},s,\bar{t},\bar{u}$"; a particular one of the k matrices thus far constructed ($k \geqq L$). Since A_L has been normalized, the first column has the largest column sum. We therefore select this column as the next branch point. Let us say that this column corresponds to the v^{th} column of A.

S6. Denoting A_L by $[\delta_{L1}, \delta_{L2}, \ldots, \delta_{Lm_L}]$, we construct the next sub-matrix, A_{k+1} ($=[\delta_{L2}, \delta_{L3}, \ldots, \delta_{Lm_L}]$), thus finding the sub-matrix

corresponding to a decision to exclude column one of A_L (column v of A) from the α-set. A_{k+1} is of size m_{k+1} by n_{k+1}, and A_{k+1} $\varepsilon\, \mathfrak{A}(R_{k+1}, S_{k+1})$.

S7. Noting that columns p, q, and s of A have been included up to this stage, we form a vector of row sums of included columns, which we shall call RS. That is, referring back to the A matrix, we compute for each row, the number of ones in columns p, q, and s. Obviously, for this particular label, the sum cannot exceed three. Now we compute a test vector, RT. Let the i^{th} component of RT be the maximum of zero and $c_i - rs_i$ (the i^{th} component of RS). The vector, RT, gives us the number of ones yet to be included in each row of the matrix, by some subsequent choice of columns. Since both RS and RT are vectors which are required only at this branch, and will thence be discarded, there is no need to subscript them.

S8. Returning to our decision to exclude column one of A_L, we examine each component of R_{k+1}. If any component of R_{k+1} is less than its corresponding component of RT, it is infeasible to exclude this column. We set $\varepsilon'_{k+1} = \infty$, but retain the label of A_{k+1} which is, in this case, "$p,q,\bar{r},s,\bar{t},\bar{u},\bar{v}$". We then proceed to the next step. If, on the other hand, we determine that the label represents a feasible set of columns, that is, no component of R_{k+1} is less than its corresponding component of RT, we compute ρ_{k+1} by equation (2-5) using the vector RT in place of C. Since there are three unbarred columns in this typical label, we set $\varepsilon'_{k+1} = \rho_{k+1} + 3$.

S9. We now construct matrix A_{k+2} with label "$p,q,\bar{r},s,\bar{t},\bar{u},v$". This matrix corresponds to the decision to include column v of A in the α-set. We delete column one of A_1. For every row which had a

"1" column one of A_L, we subtract one from the appropriate component of RT. If this component now is zero, we delete the corresponding row of A_L. When the procedure is completed, we have the matrix A_{k+2}. We compute R_{k+2} and S_{k+2}; and finally, ε'_{k+2} which equals $\rho_{k+2} + 4$ in this case. If, however, either dimension of the matrix becomes zero at this step, we proceed to step S11, as it is possible that this represents termination. ρ_{k+2}, of course, is computed using the vector RT instead of C.

S10. Let P be the k+2 dimensional vector whose components are the ε'_i. We find the minimum component of P, choosing arbitrarily in case of a tie, and use this component's corresponding matrix for our next path. Although an arbitrary choice in case of a tie will lead to solution, there are two techniques for choosing between branches that will probably shorten the algorithm somewhat. These are, either to stay with the current branch in case of a tie in which the current branch is involved; or to take the branch which has the largest number of unbarred columns in its label. The second method is probably the best, but in the computer algorithm we shall use neither technique; branching instead on the matrix with the smallest sub-script because of programming simplicity. Let us say we have chosen matrix A_L for our branching matrix. We set $\varepsilon'_L = \infty$, and return to step S5, continuing the algorithm.

S11. Since A_{k+2} is of zero dimension; $\rho_{k+2} = 0$. Then ε'_{k+2} is equal to the number of columns that are unbarred in the label of A_{k+2}. Now if $\varepsilon'_{k+2} > \varepsilon'_i$ for any $i < k+2$, we have not <u>necessarily</u> found a solution, so we return to step S10, after duly recording the proper values for all of the parameters associated with this sub-matrix.

Of course, it is now meaningless to consider row sum and column sum vectors. This is a minor point, since the only purpose of these vectors is in computing ρ, and in deriving subsequent matrices. It is clear, though, that if we at a later time choose to branch upon this matrix of zero dimension, it is because we have found that it is after all, an optimal solution to our problem.

If, on the other hand, $\varepsilon'_{k+2} \leqq \varepsilon'_i$ for all $i \leqq k+2$, the unbarred columns in the label of A_{k+2} constitute a minimal α-set of representatives for A, and the cardinality of this set of columns is the α-width of A. Thus we have arrived at a termination point of the algorithm. In the next sub-section, we shall prove that the algorithm does find a minimal α-set of representatives, and that it terminates in a finite number of steps.

3.3 Proof that a Solution is Reached.

We need to show that the algorithm does find a minimal α-set of representatives even though many possible combinations of columns have not been considered. It is first necessary, though, to show conditions under which the α-width exists. We have already stated, in step P1, that if $\alpha < r_m$, the α-width does not exist. We now prove a necessary and sufficient condition for the existence of the more general C-cover of A:

Theorem 3.1.

The matrix, A, has a C-width for every vector, C, whose components, c_i are bounded above by r_i.

Proof: By hypothesis, $c_i \leqq r_i$; hence A, itself, is a C-cover for every admissible vector, C. For a fixed C, the collection of all

C-covers is thus non-empty, and clearly is finite. Then the collection has a minimal member, and the cardinality of this minimal member is the C-width of A.

QED

Corollary:

The matrix, A, has an α-width, $\varepsilon(\alpha)$, for each integer α in the interval, $1 \leqq \alpha \leqq r_m$.

We shall now demonstrate that the branch and bound technique of section 3.2 will find the minimal C-cover of A in a finite number of iterations. We shall further show, that the branch and bound technique is independent of the technique for computing a bound on ε, under some rather simple restrictions. We shall call $\tilde{\varepsilon}(C)$, the C-width of the class, $\mathcal{U}(R,S)$. Now $\tilde{\varepsilon}(C)$ is clearly a function of the dimensions of the matrices in the class. Let $\rho*(C)$ be an estimate of $\tilde{\varepsilon}(C)$ such that $\rho*(C) \leqq \tilde{\varepsilon}(C)$, and such that for the class $\mathcal{U}$, one of whose dimensions is zero, $\rho*(\vec{0}) = \varepsilon(\vec{0}) = 0$; where $\vec{0}$ is the m dimensional zero vector. We insist in what follows that the estimating technique for computing $\rho*(C)$ be applied consistently. The parameter, ρ, described in section two satisfies the above requirements on $\rho*(C)$.

Let A be a given matrix and estimate $\tilde{\varepsilon}(C)$ by $\rho*(C)$. Then the C-width of A is not less than $\rho*(C)$. Now construct matrices A_1 and A_2 as in steps S6 and S9 of section 3.2 using any column of A, say column t, instead of that column whose sum is the largest. Estimate $\tilde{\varepsilon}(C_i)$ for each of the sub-matrices thus constructed by $\rho_1^*(C_1)$, and $\rho_2^*(C_2)$ respectively. Then the C_1-width of A_1 is not less than $\rho_1^*(C_1)$,

and the C_2-width of A_2 is not less than $\rho^*_2(C_2)$. The vectors, C_i, have components equal to the number of "1"'s yet necessary to represent the i^{th} row of A. For example, if the i^{th} component of C were 2 and the selected column contained a one in its i^{th} place, then the i^{th} component of C_1 would be 2; but the i^{th} component of C_2 would be 1. Now since column t of A must be either included in, or excluded from the minimal C-cover of A, the C-width of A is not less than $\min[\rho^*_1(C_1), \ \rho^*_2(C_2)+1]$. We need no longer consider $\rho^*(C)$ as an estimate of the C-width of A. Clearly then, if we repeat this estimating process, using A_1 or A_2 as a new given matrix according to whether $\rho^*_1(C_1)$ or $\rho^*_2(C_2)+1$ is the smaller, we may compute two additional estimates of the C-width of A. Eventually (after a finite number of such estimates have been made), we shall construct a matrix, one of whose dimensions is zero. In that case, $\rho^*_{2k}(C_{2k}) = 0$, and the C-width of A cannot be less than the cardinality of the set of columns slated for inclusion in the C-cover of A. This set of columns is, in fact, a C-cover, and if the cardinality of this set is less than or equal to all of the other computed estimates of the C-width of A, then it is a minimal C-cover, since we required that any estimate be bounded above by $\tilde{\varepsilon}(C)$.

We refer the reader once again to the scheme illustrated in Figure 5. If each branch of this tree were to be taken to its termination (at worst, the point at which each column of A had been tested either for inclusion or exclusion), each such terminal could be represented by an n-tuple as follows: let the i^{th} component be one if the i^{th} column had been included on this branch, and let it be zero otherwise. There are 2^n unique n-tuples, hence at most 2^n

corresponding terminals, each attainable in a finite number of steps. Hence the algorithm must terminate in a finite number of steps.

4. Manual Computation with the Algorithm.

Let us return to the targeting problem described in section one, and solve this problem to illustrate the use of the branch and bound algorithm. We reproduce the matrix of Figure 3, as Figure 7 for ready reference. Zeros have been suppressed, and we have appended the components of R and S to the right and bottom of the matrix, respectively. Figure 8 depicts the normalized matrix, A. We have appended the original column subscripts above the matrix.

In this example, $\alpha = 1$; and it should be noted that in general, increasing α, significantly increases the complexity of the manual algorithm, because of short-cuts used in deciding which rows may be deleted. These short-cuts are not available for $\alpha > 1$. The RT vector need not be constructed, since its components could only be zero or one, and such a simple vector can be handled by inspection. However, the short-cuts cannot be conveniently programmed, so the computer version of the algorithm can handle differing α's with almost equal facility.

	1	2	3	4	5	6	7	8	9	R
A	1	1								2
B		1	1							2
C			1	1						2
D				1	1					2
E						1	1			2
F			1				1			2
G			1					1		2
H								1	1	2
S	1	2	4	2	1	1	2	2	1	

Figure 7

	3	2	4	7	8	1	5	6	9	
	1	2	3	4	5	6	7	8	9	R
A		1								2
B	1	1								2
C	1		1							2
D			1				1			2
E				1				1		2
F	1			1						2
G	1				1					2
H					1				1	2
S	4	2	2	2	2	1	1	1	1	

Figure 8

We show, in Figures 9 and 10, the matrices A_1 and A_2, respectively, derived from A as follows: We delete column 1 from A and since this column corresponds to column 3 of the original matrix, we label A_1, "$\overline{3}$". Naturally, we have normalized both R_1 and S_1. Now locate each row of A which has a "1" in the first column. Delete this row, delete column 1, and we now have A_2, after normalizing R_2 and S_2. This criterion for deleting rows is a simplification of computing RT, which is the short-cut mentioned at the beginning of this section. The rows deleted in the example are rows B, C, G, and F. We label A_2, "3".

For the matrix, A_1, $\rho_1 = 4$, since $\sum_{i=1}^{4} s_{1i} = m_1 = 8$. Similarly, for A_2, $\rho_2 = 4$, since $\sum_{i=1}^{4} s_{2i} = m_2 = 4$. Hence $\varepsilon_i' = \rho_1 = 4$; and $\varepsilon_2' = \rho_2' + 1 = 5$.

Min $[\varepsilon'_1, \varepsilon'_2] = \varepsilon'_1$ so we choose to branch on matrix A_1. We set $\varepsilon'_1 = \infty$. Column 1 of A_1 corresponds to column 2 of the incidence matrix, and is the column which has the largest column sum.

	2	4	7	8	1	5	6	9	
	1	2	3	4	5	6	7	8	R_1
A	1				1				2
D		1				1			2
E			1				1		2
H				1				1	2
B	1								1
C		1							1
F			1						1
G				1					1
S_1	2	2	2	2	1	1	1	1	

Sub-matrix, A_1. "$\overline{3}$" $\rho_1 = 4$. $\varepsilon'_1 = 4$

Figure 9

	2	4	7	8	1	5	6	9	
	1	2	3	4	5	6	7	8	R_2
A	1				1				2
D		1				1			2
E			1				1		2
H				1				1	2
S_2	1	1	1	1	1	1	1	1	

Sub-matrix, A_2. "3" $\rho_2 = 4$. $\varepsilon'_2 = 5$

Figure 10

We delete column one of A_1 and thus have matrix A_3, which we label "$\overline{3},\overline{2}$". This sub-matrix is shown in Figure 11, below. Since row B of A_1 has row sum zero in A_3, and since there are no included columns in the label, this represents an infeasible set of column exclusions. Therefore, without further consideration, we set $\varepsilon_3' = \infty$.

	4	7	8	1	5	6	9	
	1	2	3	4	5	6	7	R_3
D	1				1			2
E		1				1		2
H			1				1	2
A				1				1
C	1							1
F		1						1
G			1					1
S_3	2	2	2	1	1	1	1	

Sub-matrix A_3. "$\overline{3},\overline{2}$" $\varepsilon_3' = \infty$

Figure 11

Next, we delete each row of A_1 which has a "1" in the first column, namely, rows A and B; and we delete the first column of A_1. This gives us matrix A_4, depicted on the next page in Figure 12. Of course, the label for A_4 is "$\overline{3}$,2". Now $\sum_{i=1}^{3} s_{4i} = m_4 = 6$; so $\rho_4 = 3$, and $\varepsilon_4' = \rho_4 + 1 = 4$. Note that the sum of column 4 of A_1 goes to zero in A_4, so we may delete it.

Now min $[\varepsilon_1', \varepsilon_2', \varepsilon_3', \varepsilon_4'] = \varepsilon_4' = 4$. Hence we choose to branch next on matrix A_4. We set $\varepsilon_4' = \infty$, and choose column one of A_4 for examination.

	4	7	8	5	6	9	
	1	2	3	4	5	6	R_4
D	1			1			2
E		1			1		2
H			1			1	2
C	1						1
F		1					1
G			1				1
S_4	2	2	2	1	1	1	

Sub-matrix A_4. "$\bar{3}$,2" $\rho_4 = 3$ $\varepsilon'_4 = 4$

Figure 12

Deleting this column, which corresponds to column 4 of the original matrix, produces A_5, with label "$\bar{3}$,2,$\bar{4}$". This sub-matrix is reproduced in Figure 13, below. Note that the sum of row C of A_4 has gone to zero in A_5.

	7	8	5	6	9	
	1	2	3	4	5	R_5
E	1			1		2
H		1			1	2
F	1					1
G		1				1
D			1			1
S_5	2	2	1	1	1	

Sub-matrix A_5. "$\bar{3}$,2,$\bar{4}$" $\varepsilon'_5 = \infty$

Figure 13

This means that the label represents an infeasible combination of columns, and we therefore set $\varepsilon_5' = \infty$ without further consideration of this sub-matrix.

Now we derive sub-matrix A_6 by deleting rows D and C from A_4, since each of these rows has a "1" in column one of A_4. We also delete column one of A_4 and give this sub-matrix the label, "$\bar{3}$,2,4". The matrix is presented in Figure 14 below. Since $\sum_{i=1}^{2} s_{6i} = m_6 = 4$, we have that $\rho_6 = 2$, and $\varepsilon_6' = 4$, since there are two unbarred columns in the label for A_6. Clearly $\varepsilon_6' = \min\,[\varepsilon_i']$ $(i=1,\ldots, 6)$; so we choose to continue along this branch. Thus A_6 will be our next branching matrix.

	7	8	6	9	
	1	2	3	4	R_6
E	1		1		2
H		1		1	2
F	1				1
G		1			1
S_6	2	2	1	1	

Sub-matrix A_6. "$\bar{3}$,2,4" $\rho_6 = 2$ $\varepsilon_6' = 4$

Figure 14

We derive sub-matrix A_7 from A_6 (see Figure 15) by deleting column one of A_6, corresponding to column 7 of the incidence matrix. This sub-matrix has label, "$\bar{3}$,2,4,$\bar{7}$". Once again we run into the situation of a row sum going to zero, so this sub-matrix, too, represents an infeasible combination of columns. Therefore, we set $\varepsilon_7' = \infty$, and proceed.

	8	6	9	
	1	2	3	R_7
H	1		1	2
G	1			1
E		1		1
S_7	2	1	1	

Sub-matrix A_7. "$\bar{3}$,2,4,$\bar{7}$" $\epsilon'_7 = \infty$

Figure 15

By the deletion of rows E and F from A_6, we arrive at sub-matrix A_8, which also has column one of A_6 deleted. This two by two matrix is displayed in Figure 16, below. The label is "$\bar{3}$,2,4,7", and we note that $s_{8,1} = m_8 = 2$; therefore, $\rho_8 = 1$, and since there are three unbarred columns in the label, $\epsilon'_8 = 4$. Note also that the column sum of column 6 of the original matrix has gone to zero, so we delete that column also. We see that $\epsilon'_8 = \min [\epsilon'_i]$ $(i=1, \ldots, 8)$, so we branch on matrix A_8. We remember to set $\epsilon'_8 = \infty$, and choose column one of A_8 for examination. This column corresponds to column 8 of the original matrix.

	8	9	
	1	2	R_8
H	1	1	2
G	1		1
S_8	2	1	

Sub-matrix A_8 "$\bar{3}$,2,4,7" $\rho_8 = 1$ $\epsilon'_8 = 4$

Figure 16

Deletion of column one of A_8 gives us a one by one sub-matrix which has label "$\bar{3}$,2,$\bar{4}$,7,8". This matrix represents an infeasible combination of columns since row G has vanished. Thus we set $\varepsilon'_9 = \infty$, and proceed.

We see immediately that we have reached termination, since the matrix A_{10} is of zero dimension and has label "$\bar{3}$,2,4,7,8". This means that $\varepsilon'_{10} = 4$ and that $\varepsilon'_{10} = \min\,[\varepsilon'_i]$ (i=1, ..., 10).

The 1-width of the incidence matrix is 4, and a minimal 1-set of representatives for the incidence matrix is the set of columns, (2,4, 7,8). These, of course, would be the station numbers that were to be targeted in our original problem.

5. Computation of $\tilde{\varepsilon}(\alpha)$.

We notice that for the very simple problem presented in section four, ten matrices had to be written down. The writer has observed that in hand computation, one matrix can be used for deriving only two or three sub-matrices before the paper becomes impossible to read. Even a small matrix requires a considerable amount of time to write down, especially when normalization cannot be done in one's head. In the computer version, due to limited storage space it is necessary to recompute a matrix each time it must be used, so even at high digital computer speeds, it would be desirable to reduce as far as possible the number of matrices that had to be examined.

Unfortunately, for the computation by hand, little can be done to simplify the problem, but in the case of the computer algorithm, it is possible to compute $\tilde{\varepsilon}(\alpha)$ exactly at little expense in time. Unfortunately, this computation will be useful only when $\alpha = 1$; and

hence, when the components of the RT vector of section three can be only zero or one. But this case is the one which is of greatest interest, and it is thus very worthwhile to study this computation.

There are at least two derivations possible. One, which is entirely combinatorial in nature, gives considerable insight into the class, $\mathfrak{A}(R,S)$, at the expense of being quite lengthy and not very intuitive. The interested reader is referred to Fulkerson and Ryser. [4].

We shall use a network derivation which is considerably shorter and more intuitive. The procedure for $\alpha = 1$ is outlined in [2]. It should be noted that the formula was first derived using network considerations. We require the following theorem in the network derivation to follow:

<u>Theorem 5.1.</u>

Let $A \varepsilon \mathfrak{A}(R,S)$ have α-width, $\varepsilon(\alpha)$. Then there is at least one matrix, A_ε, in $\mathfrak{A}(R,S)$ such that the first ε columns of A constitute a minimal α-set of representatives for A_ε.

Proof: Consider any matrix, $A \varepsilon \mathfrak{A}(R,S)$ with α-width, $\varepsilon(\alpha)$. Let E* be that subset of the columns of A consisting only of the members of the minimal α-set of representatives. If E* is the first $\varepsilon(\alpha)$ columns of A, then $A = A_\varepsilon$. Therefore we assume that column p is the leftmost column of A not in E*. Now locate column k such that column k is the rightmost column of A in E*.

Let $R_E = (r_{E1}, \ldots, r_{Em})$ be the vector of row sums of E*. Now if $r_{Ei} = \alpha$ and there is no $a_{ip} = 0$ for which $a_{ik} = 1$, $(i=1, \ldots, m)$, we may replace column k by column p in E*, and the new columns of E* are a minimal α-set of representatives. Suppose therefore, that we have for some i, $r_{Ei} = \alpha$, $a_{ip} = 0$ and $a_{ik} = 1$. We call a_{ik} a critical

one of E*. Then since $s_p \geq s_k$, there must be an $a_{jp} = 1$ for which $a_{jk} = 0$. ($j \neq 1$). Further, for each critical one in column k, there is a distinct one in column p with a corresponding zero in column k. We perform interchanges on such critical ones, the typical interchange resulting in $a_{ip} = 1$; $a_{ik} = 0$; $a_{jp} = 0$; and $a_{jk} = 1$. We may now replace column k by column p in E* and the new columns of E* form a minimal α-set of representatives.

Clearly this construction is possible for each column to the left of column ε, which is not in E*. Hence the construction yields $A_\varepsilon \in \mathfrak{U}(R,S)$. QED

5.1 A Supply-Demand Network.

In this section we shall consider directed networks. Let N represent the set of nodes of a network and $\mathcal{Q}$ represent the set of arcs. We denote an arc between x and y, members of N; by the ordered pair, (x,y) and assert that the notation implies the arc is directed from x to y, and is not the same arc as the one denoted (y,x). We associate with each arc in $\mathcal{Q}$, a non-negative function c(x,y) called a capacity function, and a non-negative function f(x,y) called a flow function. We associate with some nodes in N a non-negative function a(x) which may be thought of as a supply of some commodity available at node x, and we associate with some other nodes in N, a non-negative function b(y) which may be thought of as a demand for some commodity by node y.

We make use of the following shorthand notation, Let S, T, be subsets of N, and let x,y be elements of N. Then by c(S,x) we mean $\sum_{s \in S} c(s,x)$, and similarly for f(S,x). Also, by c(S,T) we mean $\sum_{s \in S} \sum_{t \in T} c(s,t)$, and similarly for f(S,T). Analagous shorthand will be used for the functions, a and b.

Now let us assume we have a class of matrices, $\mathcal{U}(R,S)$. We devise a network for this class as follows: Let there be n nodes denoted $b_1, \ldots, b_n$; with demand function, $b(b_j) = s_j$, the j^{th} component of S. Let there be m nodes denoted $a_1, \ldots, a_m$; with supply function $a(a_i) = r_i$, the i^{th} component of R. Let $B = b_j$; $A = a_i$. Let $(a_i,b_j) \varepsilon \mathcal{Q}$ for all i, j. Let $c(a_i,b_j) = 1$ for all i,j. This network has an arc capacity of one for each of the m·n elements of a matrix, $A \varepsilon \mathcal{U}(R,S)$. The commodity available at the nodes of A, and required by the nodes of B is, of course, "1"'s to distribute among these m·n elements of the associated class of matrices. We construct a flow in the network satisfying the following constraints:

(6) $f(x,N) - f(N,x) \leqq a(x) \qquad x \varepsilon A$

(7) $f(N,x) - f(x,N) \geqq b(x) \qquad x \varepsilon B$

(8) $0 \leqq f(x,y) \leqq c(x,y) \qquad (x,y) \varepsilon \mathcal{Q}$

Clearly this construction is possible. For R = 2,2,2,2) and S =(3,3,2) such a network has been constructed in Figure 17. The number by each arc is the value of the flow function for that arc. Now let us construct the corresponding matrix (Figure 18). If $f(a_i,b_j) = 1$, let $a_{ij} = 1$; if $f(a_i,b_j) = 0$, let $a_{ij} = 0$. This matrix, A, is in the class, $\mathcal{U}(R,S)$. Furthermore, each unique feasible flow corresponds to a unique matrix, $A \varepsilon \mathcal{U}(R,S)$, and conversely.

Then let us ask this question: under what conditions can we construct a flow so that $f(a_i,T) \geqq \alpha$ for each i, and for $T = \{b_i, \ldots, b_\varepsilon\}$? This flow would correspond to distributing at least α ones from each a_i to the nodes corresponding to the first ε columns of a matrix in $\mathcal{U}(R,S)$. If we can locate the smallest ε for which this flow is feasible, we shall have found $\tilde{\varepsilon}$. See Theorem 5.1.

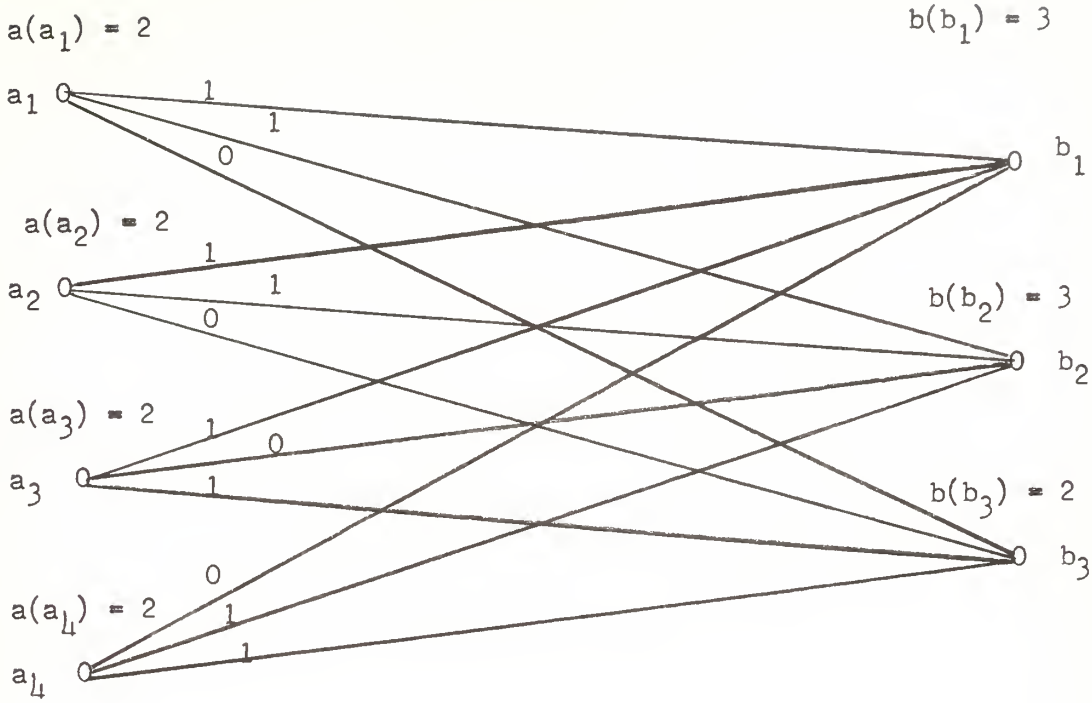


Figure 17

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

Figure 18

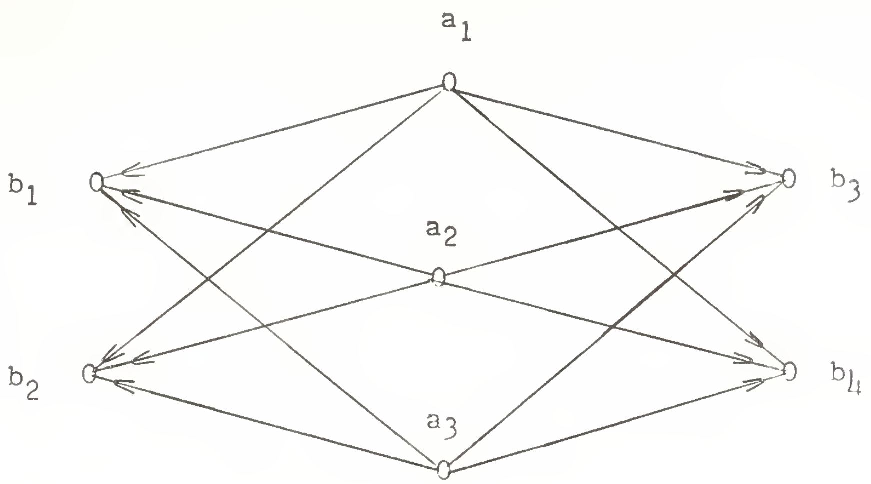


Figure 19

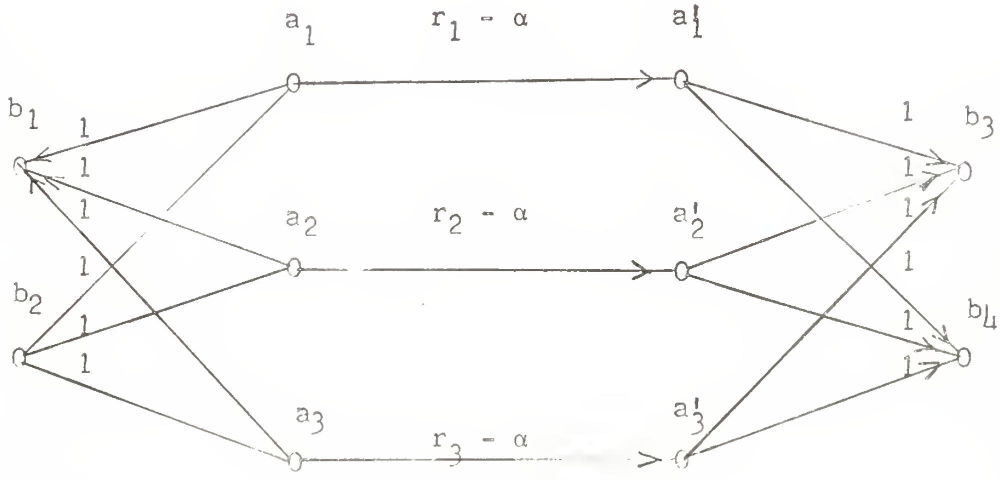


Figure 20

Since $\sum_{i=1}^{m} r_i = \sum_{j=1}^{n} s_j$; for the demands to be satisfied at each b_j, the supply at the nodes of A must be totally exhausted for a feasible flow. Consider the network of Figure 19, with four sink nodes (those with positive demand functions), and three source nodes (those with positive supply functions). Let us construct for each $\varepsilon \leqq n$; a network as follows: Let B_ε be the first ε nodes of B. Let $(x,y) \varepsilon \mathcal{Q}$ for $x \varepsilon A$ and $y \varepsilon B_\varepsilon$. Now construct m new nodes $(a'_1, \ldots, a'_m)$ and let the set of these nodes be called A'. Let $(a_i, a'_i) \varepsilon \mathcal{Q}$ for each $i \leqq m$. Let $(x,y) \varepsilon \mathcal{Q}$ for $x \varepsilon A'$ and $y \varepsilon B \sim B_\varepsilon$ (the relative complement of B_ε with respect to B). Let the following be true:

$$a(a_i) = r_i \qquad i = 1, \ldots, m$$

$$b(b_j) = s_j \qquad J = 1, \ldots, n$$

$$c(a_i, b_j) = c(a'_i, b_k) \qquad \begin{array}{l} i = i, \ldots, m \\ j = 1, \ldots, \varepsilon \\ k = \varepsilon+1, \ldots, n \end{array}$$

$$c(a_i, a'_i) = r_i - \alpha \qquad i = 1, \ldots, m$$

The construction corresponding to the network of Figure 19 for $\varepsilon = 2$ is shown in Figure 20 on the preceding page. Numbers above each arc are the capacities of the arc. What we have done is this: since the capacity of each (a_i, a'_i) is α units less than the supply at a_i, at least α units of supply must be distributed to the nodes of B_ε. We call a flow feasible if and only if constraints (6), (7), and (8) are satisfied and if

$$(9) \qquad f(x,N) - f(N,x) = 0 \qquad x \varepsilon A'$$

is also satisfied. Clearly, the smallest ε for which a feasible flow exists in this type of network is $\tilde{\varepsilon}(\alpha)$.

Theorem 5.2.

The constraints

(6) $f(x,N) - f(N,x) \leqq a(x)$ $\quad x \in A$

(7) $f(N,x) - f(x,N) \geqq b(x)$ $\quad x \in B$

(8) $0 \leqq f(x,y) \leqq c(x,y)$ $\quad (x,y) \in \mathcal{Q}$

(9) $f(x,N) - f(N,x) = 0$ $\quad x \in A'$

where $a(x) \geqq 0$, $b(x) \geqq 0$; are feasible if and only if,

(10) $b(B \cap \overline{X}) - a(A \cap \overline{X}) \leqq c(X,\overline{X})$

holds for every partition of N into subsets X and $\overline{X}$ $(= N \sim X)$.

This is the well known supply-demand theorem due to Gale. A proof may be found in [2].

We apply (10) to our network (in general) and observe that for partitions of the form:

$X = \{a_1, \ldots, a_m; a'_1, \ldots, a'_e; b_1, \ldots, b_\varepsilon; b_{f+1}, \ldots, b_n\}$ and

$\overline{X} = \{a'_{e+1}, \ldots, a'_m; b_{\varepsilon+1}, \ldots, b_f\}$

where e and f are integer parameters satisfying

$$0 \leqq e \leqq m; \qquad \varepsilon \leqq f \leqq n$$

(10) is of the form:

(11) $s_{\varepsilon+1} + \ldots + s_f \leqq (r_{e+1} - \alpha) + \ldots + (r_m - \alpha) + e \cdot (f - \varepsilon)$

The validity of the inequality is obvious except possibly for the term, $e \cdot (f - \varepsilon)$. This term is merely the number of a'_i in X, times the number of b_j in $\overline{X}$; and is the total capacity of all arcs connecting these two sets of nodes.

Theorem 5.3.

The constraints, (6), (7), (8), and (9) of Theorem 5.2 are feasible for a network of the type of Figure 20, and for fixed ε, if and only if (11) holds for all permissible values of e and f.

The proof of this theorem consists of looking at all subsets of nodes not of the form on the preceding page, and verifying that Theorem 5.2 is valid for these subsets if it is valid for the subsets of the above form. Since the proof is not very interesting, and rather lengthy, it is omitted.

Theorem 5.3 assures us that we need not test all subsets of nodes with (10) in order to assure ourselves that we have a feasible flow. Now let us multiply (11) by minus one; and rearrange terms: We arrive at:

(12) $$r_{e+1} + \dots + r_m - (s_{\varepsilon+1} + \dots + s_f) + e \cdot (f - \varepsilon) \geqq \alpha \cdot (m - e)$$

We thus have the condition that $\tilde{\varepsilon}$ is the smallest integer for which (12) is satisfied for all integer values of e and f in the ranges:

$$0 \leqq e \leqq m \qquad \varepsilon \leqq f \leqq n$$

Now the left side of (12) is the class invariant which Fulkerson and Ryser call $N(\varepsilon,e,f)$. [4]. For ease of computation we define a function, $Q(\varepsilon,e,f) = N(\varepsilon,e,f) - \alpha \cdot (m - e)$. $\tilde{\varepsilon}$ is the smallest ε for which

(13) $$Q(\varepsilon,e,f) \geqq 0 \qquad 0 \leqq e \leqq m; \qquad \varepsilon \leqq f \leqq n$$

We take first differences with respect to ε, e and f; and derive the following recursion formulas:

(14) $$Q(\varepsilon+1,e,f) = Q(\varepsilon,e,f) + s_{\varepsilon+1} - e$$

(15) $$Q(\varepsilon,e+1,f) = Q(\varepsilon,e,f) + f + \alpha - \varepsilon - r_{e+1}$$

(16) $$Q(\varepsilon,e,f+1) = Q(\varepsilon,e,f) + e - s_{f+1}$$

Since we know that ρ is a lower bound on the α-width of any $A \in \mathfrak{A}(R,S)$ we may take $\varepsilon = \rho$, and compute an $m+1$ by $n-\varepsilon$ array making liberal use of (15) and (16). If one of the numbers is negative, we increment ε by one using (14), and compute the $m+1$ by $n-\varepsilon$ array

for this new value of ε. When an array is found which contains no negative members, we have found $\tilde{\varepsilon}$.

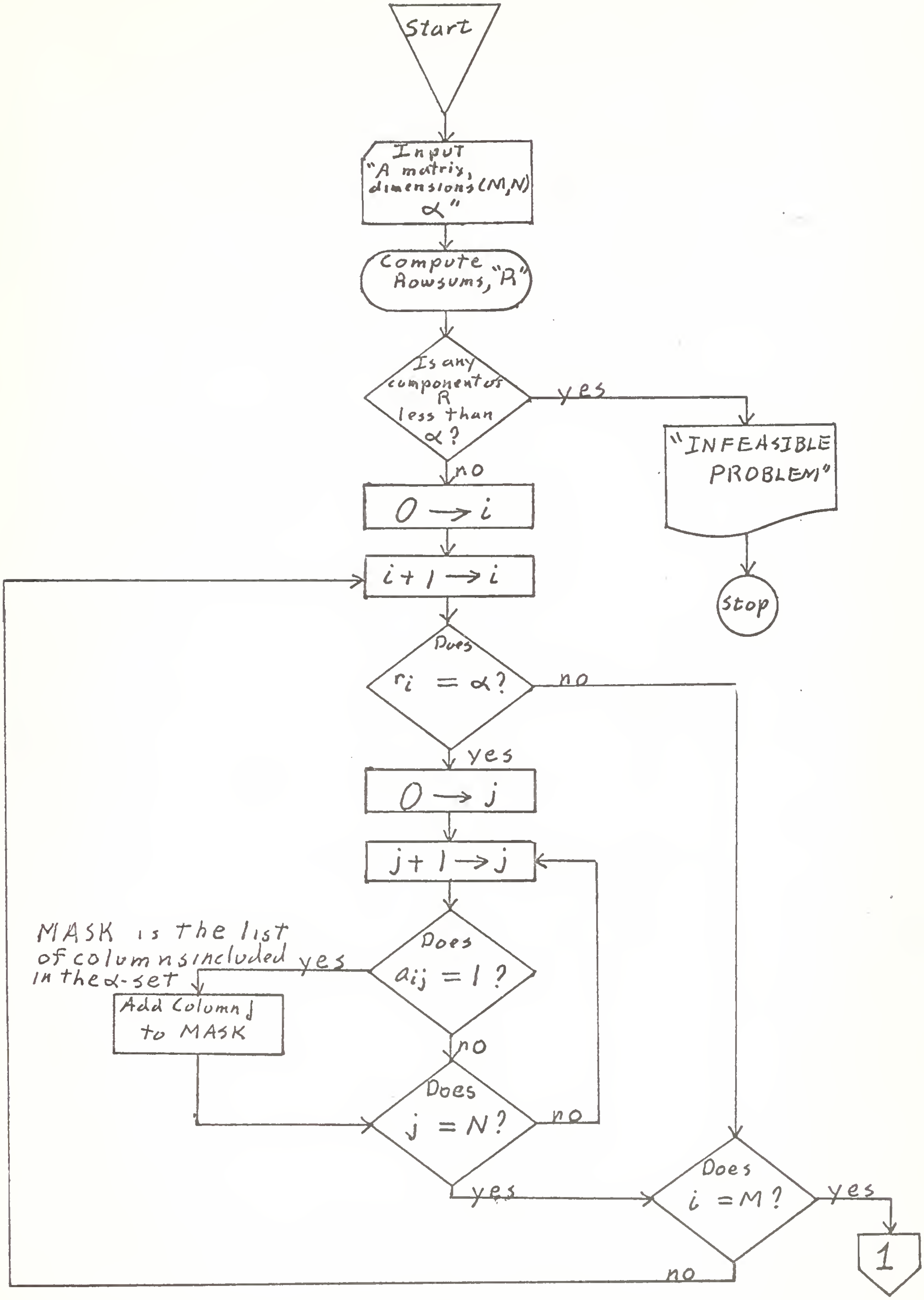
Now, as we mentioned before, we do not advise this procedure for hand computation, and it cannot be used for α greater than one, but the case $\alpha = 1$ is the most important case by far, as will be seen in section seven, and a digital computer is admirably suited to perform these simple arithmetic computations. In the next section we shall discuss the computer program in which the above formula was used; and the results obtained with a large number of matrices.

6. The Algorithm Program.

We present a procedural flow chart for the branch and bound algorithm in Figure 21. As much as possible of the procedure is described in abbreviated, but intuitive language. Where variable names are necessary they are either the names given to the same variables used in Sections 3.1 and 3.2, or they are defined on the flow chart itself near the point at which they are first used. Variable names are used that are in reasonable agreement with the corresponding names used in the computer program.

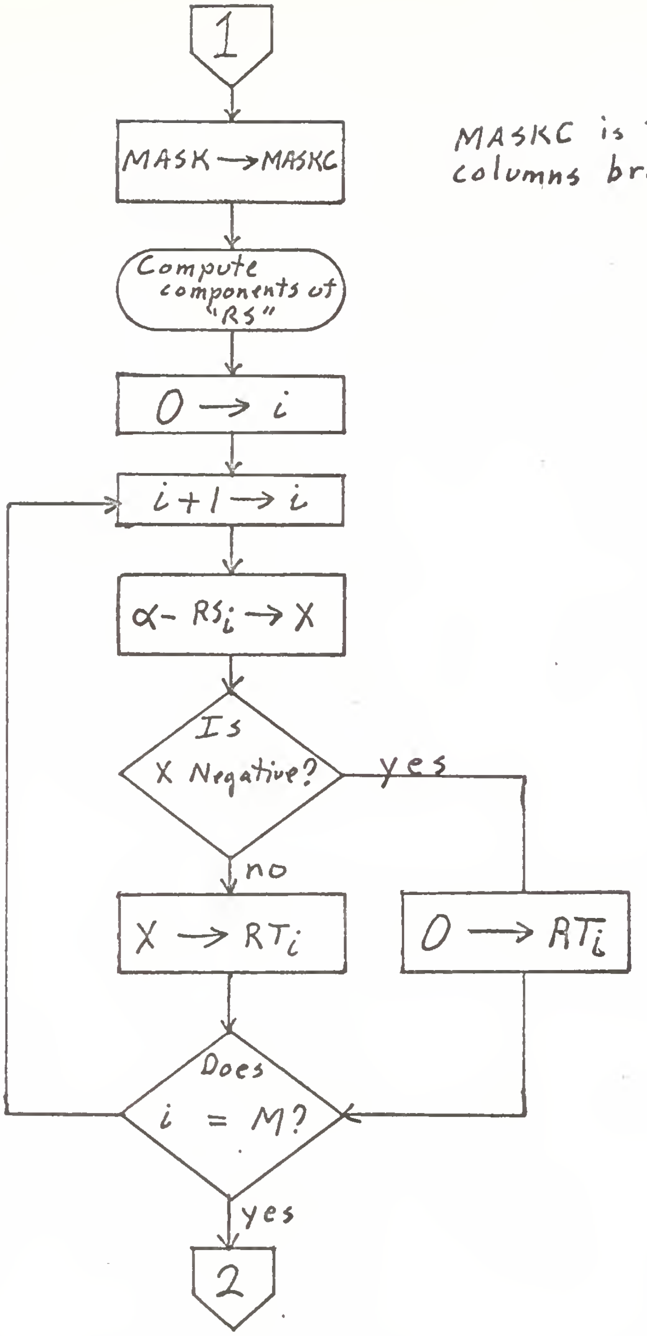
The algorithm was programmed for the Control Data 1604 computer using FORTRAN 63 source language, and CODAP 1 assembly language. The assembled program is included in this paper as Appendix I. There are several features of the program which deserve discussion.

Since we are dealing with matrices composed of zeros and ones, storage space can be conserved by letting a single bit represent an element of the matrix. Then we may use logical operations to manipulate the matrix. Normally this would require writing the entire program at

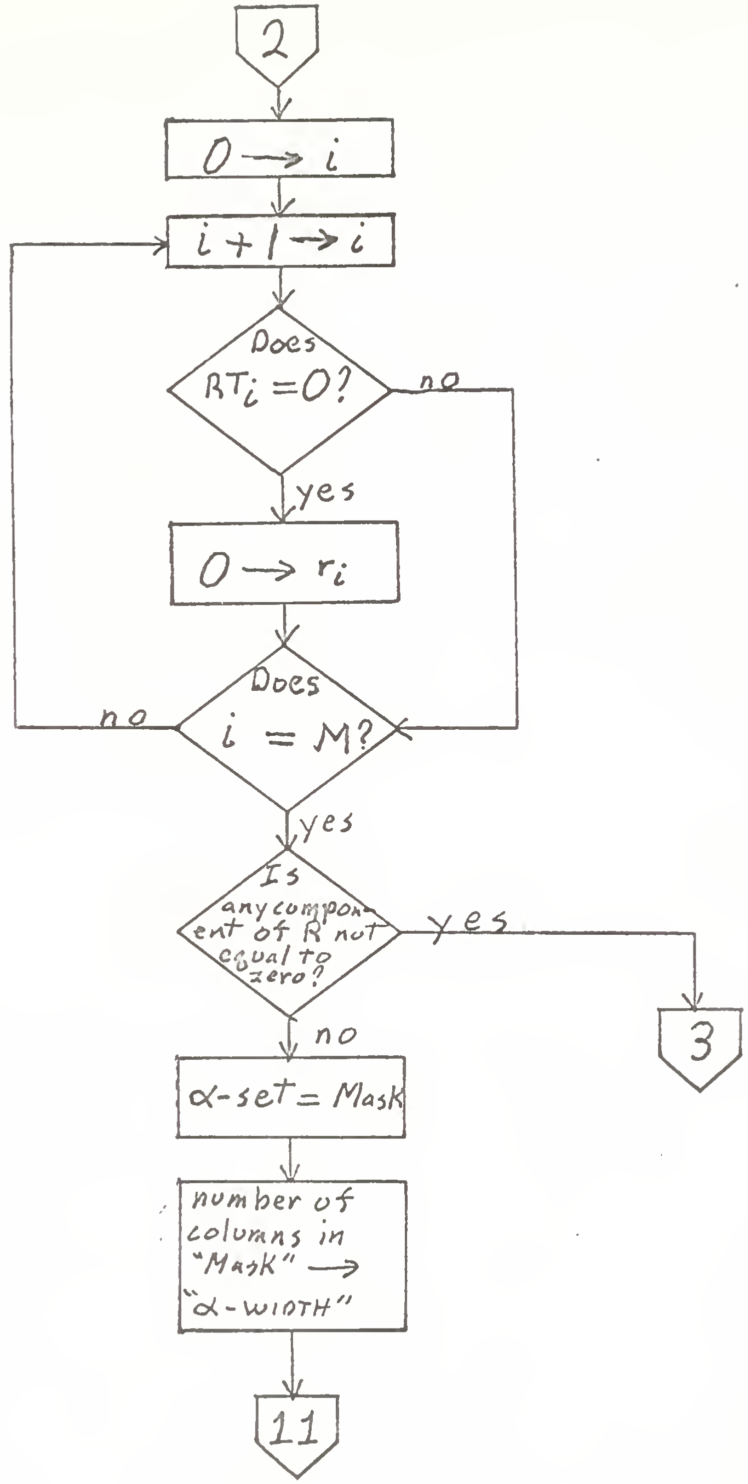


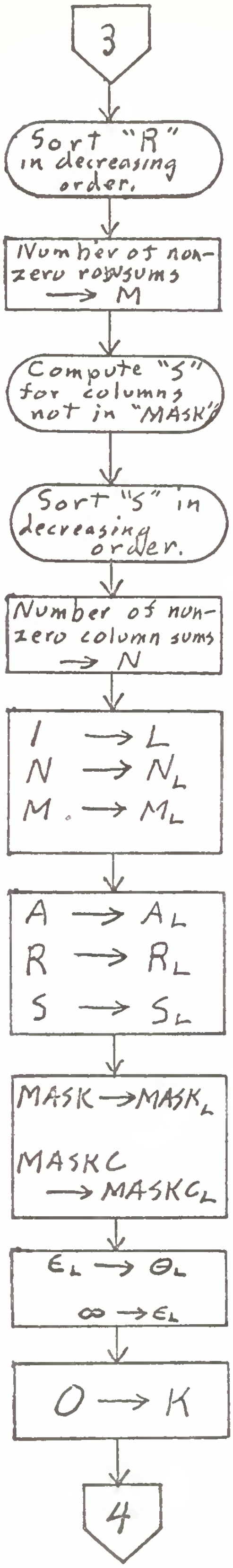
Start
Input "A matrix, dimensions (M,N) α"
Compute Rowsums, "R"
Is any component of R less than α?
yes
"INFEASIBLE PROBLEM"
Stop
no
0 → i
i + 1 → i
Does r_i = α?
no
yes
0 → j
j + 1 → j
Does a_ij = 1?
yes
MASK is the list of columns included in the α-set
Add Column j to MASK
no
Does j = N?
no
yes
Does i = M?
yes
1
no

Figure 21

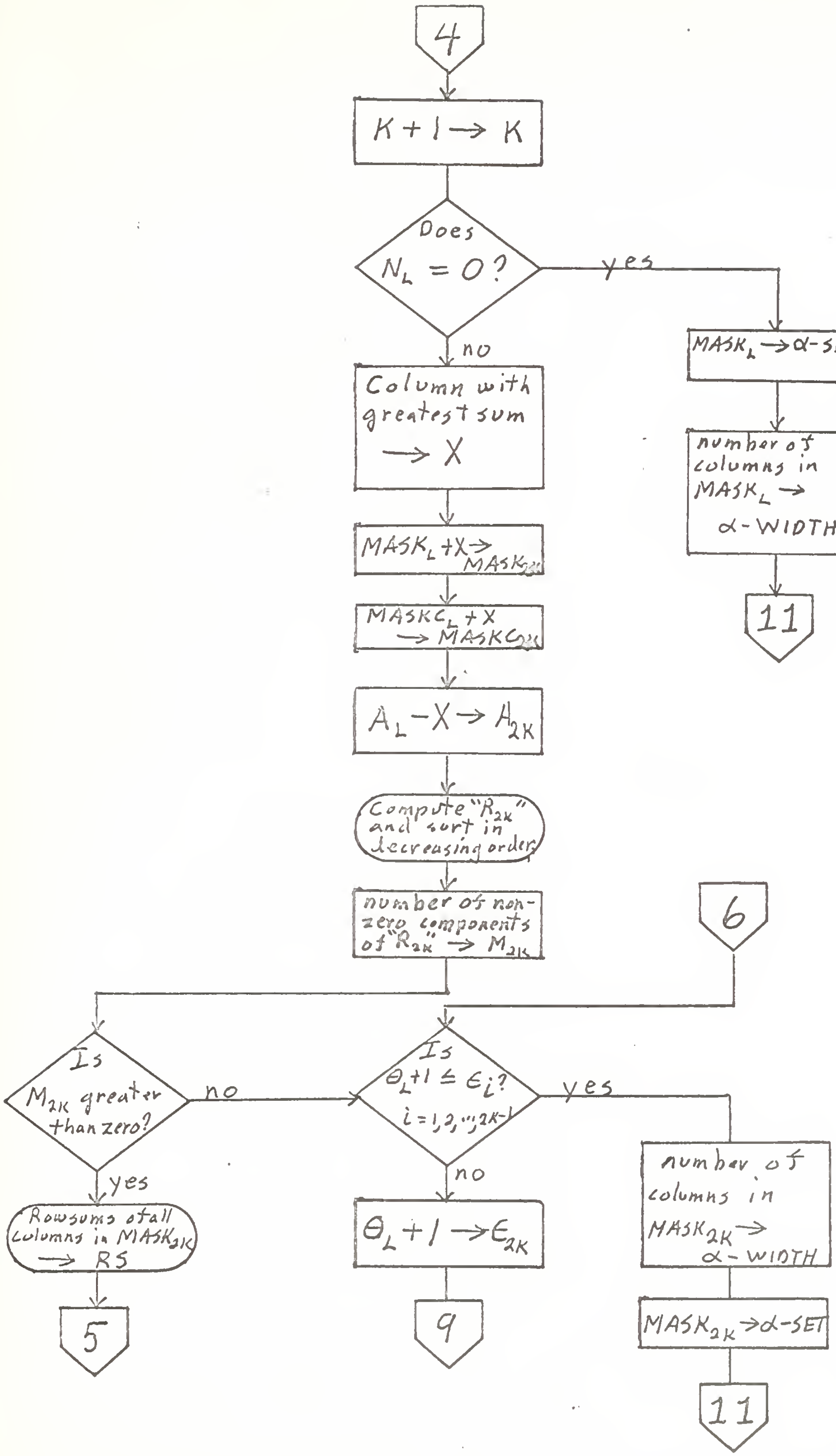


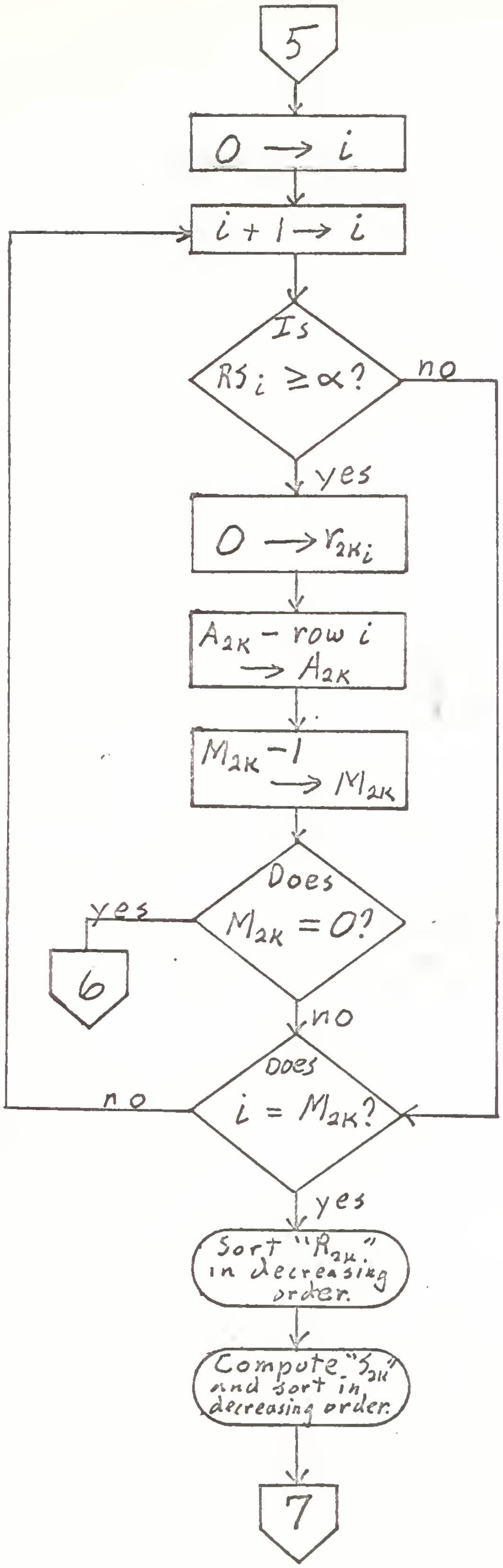
MASKC is the list of columns branched upon.

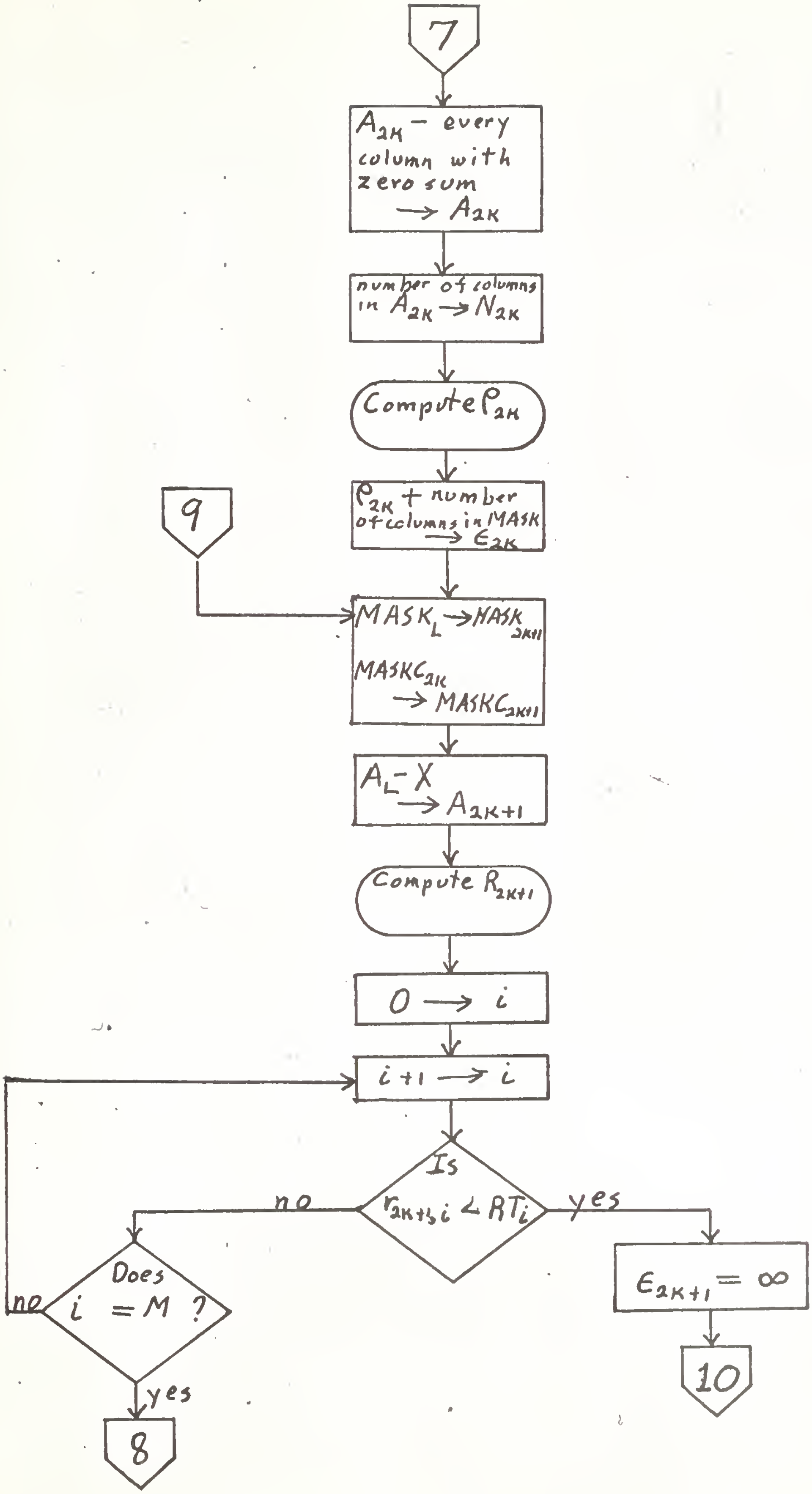

2
0 → i
i+1 → i
Does $RT_i = 0$?
no
yes
$0 \to r_i$
Does i = M?
no
yes
Is any component of R not equal to zero?
yes
3
no
α-set = Mask
number of columns in "Mask" → "α-WIDTH"
11

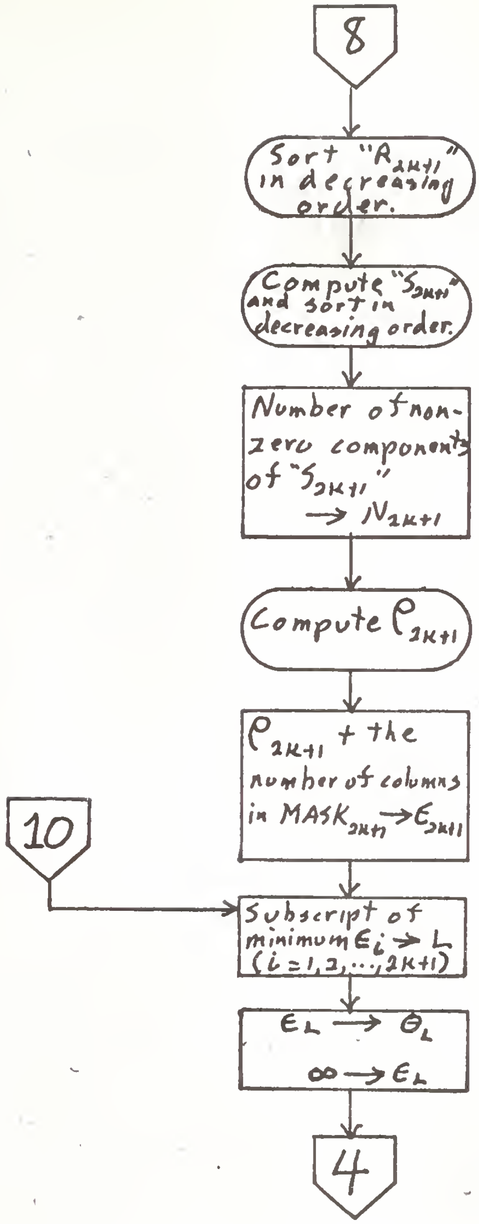


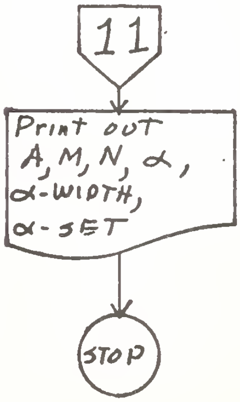
L is the index of the branching matrix


4
K + 1 → K
Does
N_L = 0?
yes
no
MASK_L → α-SET
number of
columns in
MASK_L →
α-WIDTH
11
Column with
greatest sum
→ X
MASK_L + X →
MASK_2K
MASKC_L + X
→ MASKC_2K
A_L − X → A_2K
Compute "R_2K"
and sort in
decreasing order
number of non-
zero components
of "R_2K" → M_2K
6
Is
M_2K greater
than zero?
no
yes
Rowsums of all
columns in MASK_2K
→ RS
5
Is
θ_L+1 ≤ ε_i?
i = 1,2,…,2K-1
yes
no
θ_L + 1 → ε_2K
9
number of
columns in
MASK_2K →
α - WIDTH
MASK_2K → α-SET
11


5
0 → i
i + 1 → i
Is RS_i ≥ α?
no
yes
0 → r_2K_i
A_2K − row i → A_2K
M_2K − 1 → M_2K
Does M_2K = 0?
yes
6
no
Does i = M_2K?
no
yes
Sort "R_2K" in decreasing order.
Compute "S_2K" and sort in decreasing order.
7


7
A2K - every column with zero sum → A2K
number of columns in A2K → N2K
Compute P2K
P2K + number of columns in MASK → E2K
9
MASKL → MASK2K+1
MASKC2K → MASKC2K+1
AL - X → A2K+1
Compute R2K+1
0 → i
i+1 → i
Is r2K+1,i < RTi
no
yes
Does i = M ?
no
yes
E2K+1 = ∞
8
10


8
Sort "R_{2K+1}" in decreasing order.
Compute "S_{2K+1}" and sort in decreasing order.
Number of non-zero components of "S_{2K+1}" → N_{2K+1}
Compute ρ_{2K+1}
ρ_{2K+1} + the number of columns in $MASK_{2K+1}$ → E_{2K+1}
10
Subscript of minimum E_i → L ($i = 1, 2, \cdots, 2K+1$)
$E_L \rightarrow \theta_L$
$\infty \rightarrow E_L$
4


11
Print out A, M, N, α, α-WIDTH, α-SET
STOP

the assembly language level; but we avoid this by taking advantage of a capability of FORTRAN 63 which permits the programmer to define his own type of arithmetic.

The CDC 1604 word size is 48 bits. We chose to write the program to accept matrices up to dimensions 144 by 144. We store a single row of the matrix in three consecutive computer words; hence an entire matrix requires only 432 words of storage. The first word of row i contains elements a_{i1} through a_{i48}; the second word contains a_{i49} through a_{i96}; and the third word contains a_{i97} through a_{i144}.

We define, according to the rules of FORTRAN 63, a TYPE LOGIC5 arithmetic in which an elemental word consists of three consecutive words of memory. We call such an elemental word a TYPE LOGIC5 word. Thus, one LOGIC5 word is equivalent to one entire row of a matrix, or other variable which needs to be three computer words in length. For instance, we shall require several masks with which to derive the various sub-matrices, and each such mask must consist of three computer words.

We shall need to take logical sums and products, to complement words, to clear words of ones in certain bit positions; and we shall require a method of generating a 1 in any of the 144 bit positions of a LOGIC5 word. We define, through the subroutine, Q1QMATH, the symbol "+" to mean logical sum; the symbol "*" to mean logical product; the symbol "-" with two arguments to mean "set the i^{th} bit of the first argument to zero if the i^{th} bit of the second argument is one"; and the symbol "-" with one argument to mean "complement the argument". We also define "ARGUMENT /j" to mean "set the j^{th} bit of the argument to one, and all other bits to zero, counting from the leftmost bit position of the argument".

The only requirement we have for generating the sub-matrices of the given matrix is that we must be able to compute the corresponding row sums and column sums for use in estimating $\tilde{\varepsilon}$. We may compute the row and column sums in the computer program without deriving each of the sub-matrices through the use of suitable masks. We require two such masks; one is a mask of columns upon which the program has already branched; and the other is a mask of columns chosen for inclusion in the minimal α-set at the current branch. In each case, a 1 in the i^{th} bit position of a mask indicates that column i is a member of the set of columns which the mask represents.

Almost all arguments used in the various subroutines are stored in COMMON. This decreases the computation time at the expense of requiring difficult to follow indexing of the parameters. Most such parameters are stored in an array, IDATA. This array is really three consecutive arrays of parameters associated respectively with the matrices A_L, A_{2K}, and A_{2K+1} of Figure 21. The correspondence between IDATA and the mnemonic variable names may be found in the EQUIVALENCE statement near the beginning of the program.

The masks and bound of all matrices must be retained in storage, but other parameters, (row sums, column sums, dimensions, etc.) are recomputed each time they are required. If a random access storage device (such as a magnetic disc) is available a savings of computation time would result from the storage of these parameters.

Up to 2000 sets of parameters can be retained in core storage simultaneously. When this limit is reached, the section of the program from statements 192 to 193 searches for any sets of parameters no longer required, discards them and compresses the remaining parameters into the

```
A MATRIX OF DIMENSION (  8) BY (  9).


60C0C0000000CCO                    C                 C
30CC0000C0000CCO                   C                 C
140000000C0CCCCO                   0                 0
 6CC000C00000CCO                   C                 C
 14C00000C000CCO                   C                 0
104CC0000C0000CC                   C                 C
102C0000CC0000COO                  0                 C
  30C000000000CCC                  C                 C

        1-WIDTH OF THE MATRIX IS   4.


NUMBER OF ITERATIONS REQUIRED-    13
TIME FOR COMPUTATION-  OMIN, 1SEC.

COLUMNS IN MINIMAL REPRESENTATIVE SET ARE-
             2,    4,    7,    8,
```

Figure 22

front of the storage area. This effectively increases storage space up to the point at which there are 2000 current branches of the algorithm. (Current branches are those branches for which the corresponding estimate of ε is less than "infinity".)

Sample output is shown in Figure 22. This matrix can be recognized as the incidence matrix of the communications network discussed in Section one. Note that the matrix is printed in octal format which must be converted by hand to the proper (0,1) form. Each digit of the output represents three elements of the matrix; for example, the digit "5" represents the three elements "1, 0, 1".

Short, but descriptive comments separate major sections of the program listing by tasks, and introduce each of the subroutines. The various CDC 1604 instruction manuals and programming manuals may be consulted for further information.

The program is not very efficient in its present form: many programming conveniences such as the use of TYPE LOGIC5 arithmetic, and the use of subroutines, makes writing of the program simpler at the expense of generating many otherwise unnecessary instructions. As a first step toward improving the efficiency, the author recommends elimination of TYPE LOGIC5 arithmetic, substituting in its place, CODAP1 subroutines to perform the necessary substitute operations, and using direct calls to these subroutines in place of the operations symbols. In addition, it is recommended that all present subroutines written in FORTRAN 63 be incorporated into the main program. Program space is not critical in a computer the size of the CDC 1604, and by writing the subroutines as part of the main program, advantage may be taken of task specialization. For instance, subroutine ROWSUM computes

the sum of all M rows of the matrix each time it is called. It takes as much time to compute a row sum which is zero as one which is not; but we have information which could be used to specialize the routine so that it skips over rows whose sum is zero.

A still better technique would be to write the entire program at the assembly language level; especially if the user intends to use the program for more than the solution of a few matrices.

6.1 Results of Using PROGRAM WIDTH.

If a matrix has α-width, ε, and we were to attempt to find the α-width by looking at all possible sets of α columns, then all possible sets of α + 1 columns, and so forth up to all possible sets of ε − 1 columns and finally some sets of ε columns, we should have to look at X sets of columns for

$$(17) \qquad \sum_{k=\alpha}^{\varepsilon-1} \binom{n}{k} + 1 \leqq X \leqq \sum_{i=\alpha}^{\varepsilon} \binom{n}{k}$$

We should have to look at this number of sets of columns using the branch and bound algorithm also, if all of the estimates of ε which were current turned out to be equal. It is conceivable that this could happen for some problem; hence we must take (17) as an upper bound on the number of branches which must be investigated by the program. Now the branch and bound algorithm is not the most efficient way to search subsets of columns, so we are quite interested in determining just how far below the upper bound we can stay by branching and bounding.

Since we cannot express any theory to demonstrate the efficiency of the algorithm, the only choice open to us was to solve many problems of varying sizes in hopes that trends could be established. It is for this reason that subroutines RANDOM and RANDGEN were added to the program. These two subroutines generate matrices of any size up to 144

by 144. A uniform random number generator is used to generate three consecutive random numbers which represent one row of a matrix of 144 columns. If a matrix of N columns is desired (N < 144) bit positions N + 1, ..., 144 of the three word element are set to zero. The number of ones remaining in the three words is computed and compared to a user supplied argument, NONES. If NONES is less than the remaining ones in the three word element, another set of three words is generated, the appropriate bit positions cleared to zero, and then the logical product of the two elements is taken. This procedure is repeated until NONES is greater than or equal to the number of ones remaining in the three word element. Thus NONES represents the maximum permissible row sum of any row in the matrix. The three word element is then assigned as a row of the matrix, and the procedure is repeated until an entire matrix has been generated. We are thus reasonably sure of a random distribution of ones throughout the matrix, and we have some control over the density of ones in the matrix. Matrices of any dimension are generated in no more than a few seconds.

Our original plan was to generate and solve five matrices of each of 112 sets of dimensions for the matrix. It was felt that such a set of matrices would be a statistically significant sample from which computation time could be functionally related to such parameters as matrix dimensions. Unfortunately, time has prevented the completion of this scheme. Hence all remarks that follow in this section are without statistical significance.

We have been able to generate and solve over 200 matrices of varying dimensions for their 1-width; one being by far the most important

value for α. Dimensions of matrices generated were from the 8 by 9 problem of section one to matrices of dimensions 144 by 25, and 35 by 100. Some relatively square matrices of size 50 by 45 are included. As is to be expected, computation time varies directly with number of branches considered when matrix dimensions are held constant. Let us therefore make some remarks about the number of branches considered by the program in solving these matrices.

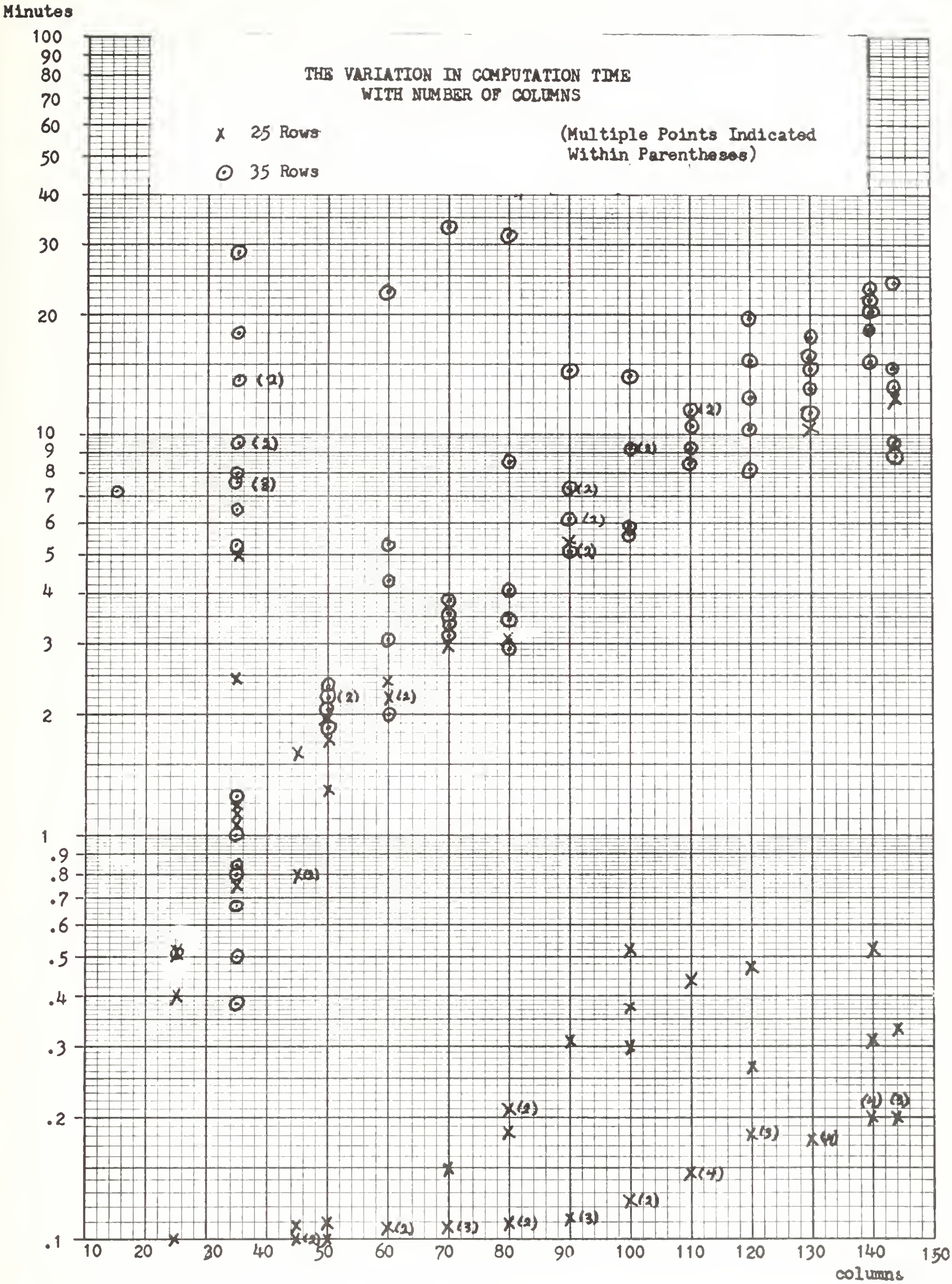
It is clear that the number of branches is a function of the number of columns and of the actual 1-width of the matrix. Not quite so obvious is that the number is a function of the number of rows in the matrix. However the fluctuations apparent in the number of branches is very wide. For instance, for one matrix 260 branches were taken while for another, 1536 were taken. Both matrices were of dimensions 35 by 35, and had a 1-width of seven. It is apparent that other factors must be involved. One such factor is the distribution of ones in the matrix. One matrix, a Steiner triple system [5], which is a matrix which among other properties has all row sums equal and all column sums equal; required investigation of 1216 branches before computing the 1-width as nine. Yet this matrix had only 35 rows and 15 columns. The symmetry of the matrix made it difficult to weed out unprofitable branches. Another matrix of dimensions 50 by 45 exceeded the capacity of the program storage after 2184 branches. In every case, however, the number of branches were below the upper limit given by (17). Values seldom exceeded 600 for any of the matrices.

Of more practical interest is the time required for computation. The CDC 1604 has an effective cycle time of 4.8 μ sec. The longest

time required to solve any problem was 36 minutes, although there were problems which had not been solved when the program was stopped by the operator after about 45 minutes. The matrix which required 36 minutes was of dimensions 50 by 45 and had a 1-width of six. The relationship between branches and computation time is rather interesting. Matrices of dimensions 125 by 25 required a little over one second per branch whereas matrices of dimensions 25 by 120 required between two and three seconds per branch. This seems to verify that advantage could be gained by eliminating the LOGIC5 arithmetic in favor of more efficient methods, since the amount of LOGIC5 arithmetic required increases with number of columns.

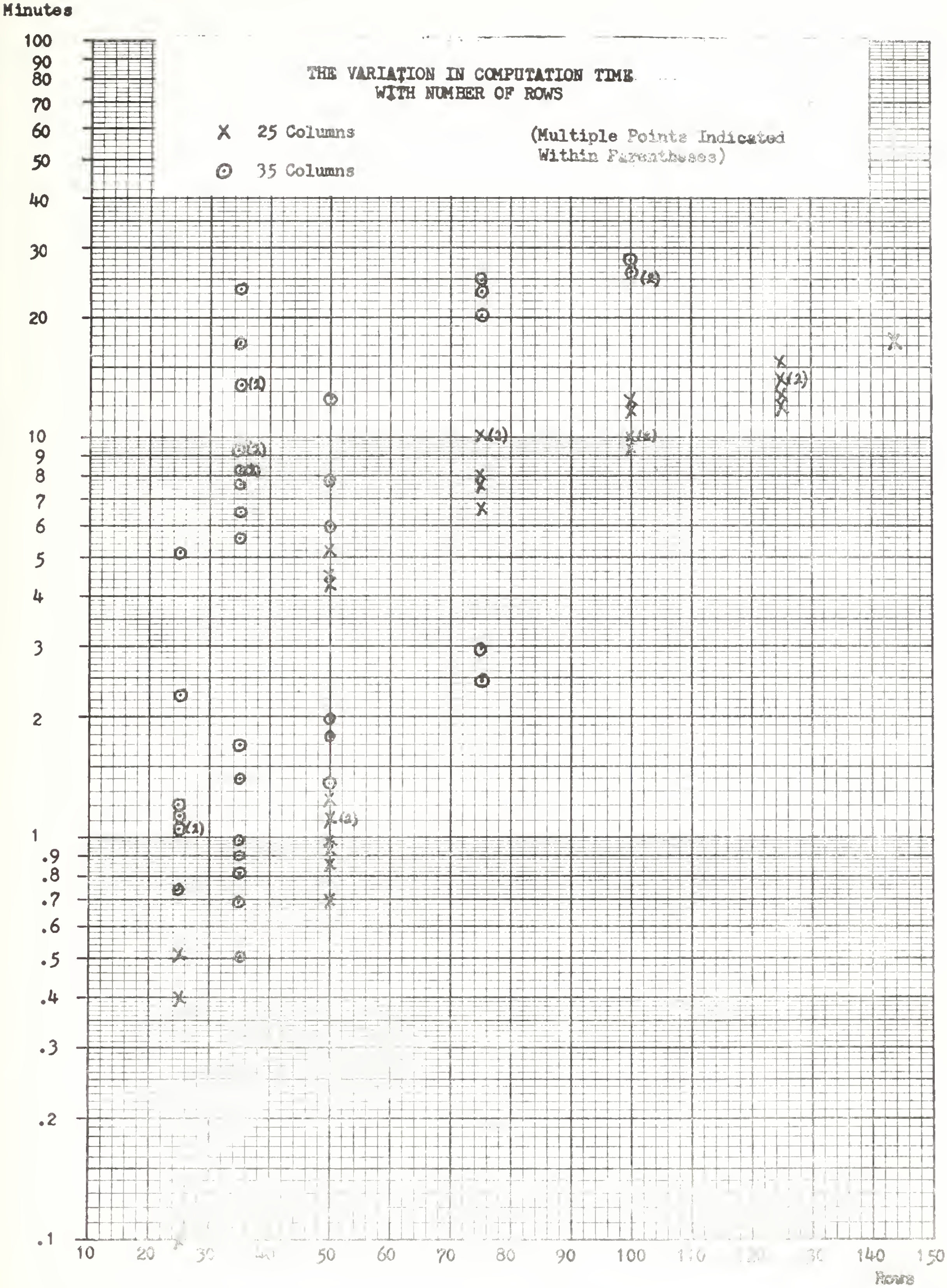
Computation time was graphed on semi-log paper versus 1) number of columns, 2) number of rows, and 3) 1-width of the matrix. Figure 23 is a graph of time versus number of columns for matrices of 25 and 35 rows. Figure 24 is of time versus number of rows for matrices of 25 and 35 columns, and Figure 25 is of time versus 1-width for matrices of dimensions 20 by 20.

From these graphs, it seems reasonable to conclude an exponential increase in computation time versus both number of rows and number of columns. No hypothesis is made about the parameters of the function. Our method of generating matrices degrades the validity of Figure 25. In order to create matrices of high 1-width, we can only lower the density of ones in the matrix. This, in turn, increases the likelihood of rows of sum one; which results in an artificial simplification of the problem. This is apparent especially in the case of the matrices of 1-widths nine and ten in Figure 25.



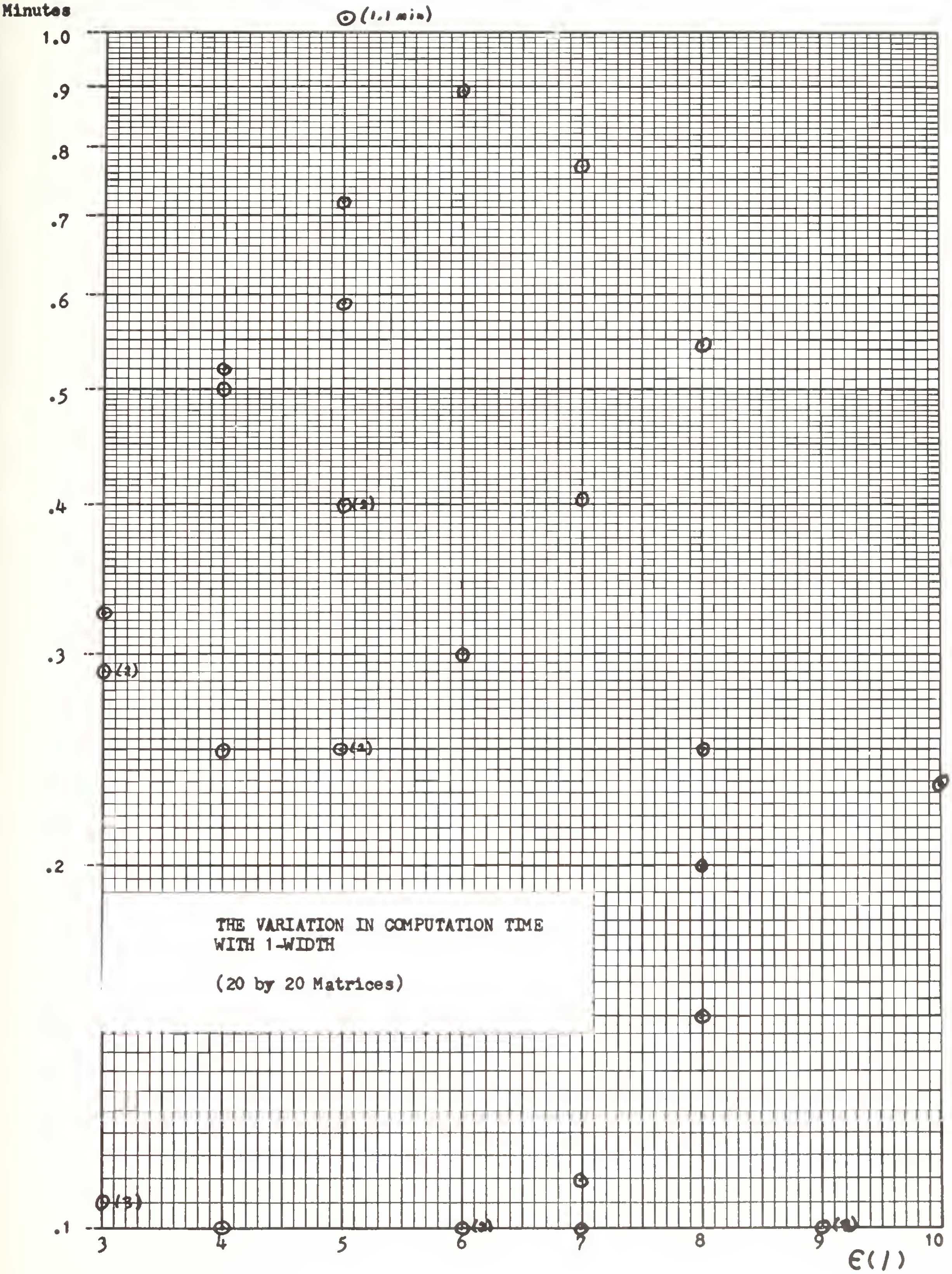
Minutes
THE VARIATION IN COMPUTATION TIME
WITH NUMBER OF COLUMNS
X 25 Rows
⊙ 35 Rows
(Multiple Points Indicated Within Parentheses)
columns

Figure 23



Minutes
THE VARIATION IN COMPUTATION TIME
WITH NUMBER OF ROWS
X 25 Columns
⊙ 35 Columns
(Multiple Points Indicated
Within Parentheses)
100
90
80
70
60
50
40
30
20
10
9
8
7
6
5
4
3
2
1
.9
.8
.7
.6
.5
.4
.3
.2
.1
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
Rows

Figure 24



Minutes
1.0
.9
.8
.7
.6
.5
.4
.3
.2
.1
3
4
5
6
7
8
9
10
ϵ(l)
(1.1 min)
THE VARIATION IN COMPUTATION TIME WITH 1-WIDTH
(20 by 20 Matrices)

Figure 25

The validity and usefulness of the algorithm has been established by the above results. Most computation times were under 20 minutes, and it is felt that computation times could be reduced more than 25% by cleaning up the program and doing without the programming convenience of TYPE LOGIC5 arithmetic.

7. Applications and Extensions.

In this final section we consider applications of the branch and bound algorithm to solution of real-world problems, and certain famous problems of the mathematical puzzle category. We shall also propose certain extensions of the algorithm as presented in section three, which enlarge the class of problems which may be solved. Certain of the problems may, indeed, be more easily solved by other methods, but they are presented here to illustrate the variety of problems which may be formulated in terms of finding the C-width of a (0,1) matrix.

7.1 The Eight Queens Problem.

A famous mathematical puzzle is the following: place the maximum number of queens on a chess board so that no two may attack each other. We construct a graph of 64 nodes, one for each square on the chessboard. Connect two nodes if a queen may move from one node to the other. The minimal 1-set of the node-arc incidence matrix is a minimal set of nodes that touch all arcs of the graph. Now since this matrix has row sums which are all equal to two, the sub-matrix consisting of all columns not in the minimal 1-set has row sums of at most one. Hence in the graph corresponding to this sub-matrix, there is no connection between any of the nodes. Thus the complement of the minimal 1-set of nodes

represents square at which the maximum number of queens may be placed so that no two may attack each other. This problem is a special case of a class of problems which also includes the targeting problem of Section one. We next present a description of this general class of problem.

7.2 The Connecting Nodes Problem.

Find the fewest number of nodes that touch all arcs in a graph. Here the rows of the incidence matrix are arcs of the graph, and the columns are nodes. The 1-width of the incidence matrix is the solution to the problem. In addition to the targeting problem of Section one, another problem of this type is the following:

Given a communications system of some type (let us say a system of highways connecting towns), what is the minimum number of arcs (highways) which must be kept safe from attack (natural disaster, etc.) so that no node is isolated? In order to solve this problem, we construct an incidence matrix as follows: list the arcs as columns and the nodes as rows. Let $a_{ij} = 1$ if the i^{th} node is a terminal of the j^{th} arc. Then the 1-width of this matrix is the solution to the problem.

7.3 Simplification of Logical Functions.

We present this problem to show the range of interpretations which may be made from the concept of α-width. The reader is cautioned, however, that the incidence matrix required for using the branch and bound algorithm to solve this problem is likely to become prohibitively large.

Given a truth table for a proposition letter formula, F, in r proposition letters, $p_1, \ldots, p_r$; find a disjunctive normal form for F which has the fewest number of terms. If we let "&" represent the conjunction operator; "+" represent the disjunction operator; and $\overline{a}$ be the negation of a, then the disjunctive normal form for a proposition letter formula is of the form:

$$(p_1 \;\&\; p_2) + (p_1 \;\&\; p_3 \;\&\; p_4) + (p_3 \;\&\; p_4) + \ldots$$

where the p_i are proposition letters. The expressions enclosed within parentheses are called terms. The problem is of interest in switching circuits and in the logical design of digital computers.

For columns of the incidence matrix take all terms having one of the forms: q_i; $q_i \;\&\; q_j$; ...; $q_1 \;\&\; \ldots \;\&\; q_r$; where q_i is either p_i, or its negation; and such that the term takes the value "true" only if F does also for all values of the p_i not explicitly present in the term. For example, if $p_1 \;\&\; \overline{p_3}$ is a term of F in three proposition letters, p_1, p_2, p_3; then both $F(p_1,p_2,\overline{p_3})$ and $F(p_1, \overline{p_2}, \overline{p_3})$ must be true if $p_1 \;\&\; \overline{p_3}$ is true. We next construct a row of the incidence matrix for each "true" entry of F in the truth table. Place a one in the column corresponding to the assignment of values to $p_1, \ldots, p_r$ which makes up the entry in the truth table corresponding to the "true" entry of F. Then place ones in all other columns which are also true for this assignment of values to the p_i. Thus if $p_1 \;\&\; p_2 \;\&\; p_3$ makes F true, a row of the matrix would have a one under this column label as well as under $p_1 \;\&\; p_2$; $p_1 \;\&\; p_3$ and so forth.

As an example, consider the truth table of Figure 26. The columns of the matrix would be labelled "$p_1 \;\&\; \overline{p_2}$"; "$p_1 \;\&\; \overline{p_3}$"; "$p_2 \;\&\; \overline{p_3}$";"$p_1\&p_2\&\overline{p_3}$" "$\overline{p_1} \;\&\; p_2 \;\&\; \overline{p_3}$"; "$p_1 \;\&\; \overline{p_2} \;\&\; \overline{p_3}$"; "$p_1 \;\&\; \overline{p_2} \;\&\; p_3$". The four rows would have ones in columns labelled as follows:

p_1	p_2	p_3	F
f	f	f	f
f	f	t	f
f	t	f	t
f	t	t	f
t	f	f	t
t	f	t	t
t	t	f	t
t	t	t	f

Figure 26

Row 1: $p_2 \ \& \ \overline{p_3}$; $\overline{p_1} \ \& \ p_2 \ \& \ \overline{p_3}$

Row 2: $p_1 \ \& \ \overline{p_2}$; $p_1 \ \& \ \overline{p_3}$; $p_1 \ \& \ \overline{p_2} \ \& \ \overline{p_3}$

Row 3: $p_1 \ \& \ \overline{p_2}$; $p_1 \ \& \ \overline{p_2} \ \& \ p_3$

Row 4: $p_1 \ \& \ \overline{p_3}$; $p_2 \ \& \ \overline{p_3}$; $p_1 \ \& \ p_2 \ \& \ \overline{p_3}$

Clearly, $p_2 \ \& \ \overline{p_3}$ and $p_1 \ \& \ \overline{p_2}$ are a minimal 1-set of representatives for the matrix, and $F = (p_1 \ \& \ \overline{p_2}) + (p_2 \ \& \ \overline{p_3})$ is a minimal disjunctive normal form.

7.4 The Minimal C-cover Problem.

It would be quite simple to extend the computer program to solve the minimal C-width problem. Essentially all that would be necessary is input revision to accept the vector, C, and the initial setting of the vector, RT to C. Of course, $\tilde{\varepsilon}$ could not be calculated, but would have to be estimated using exactly the same subroutine which is presently used for the situation, $\alpha > 1$. For an entirely different algorithm for solving the minimal C-cover problem (and hence also the α-width problem) see [8]. The minimal C-cover extension is of interest primarily as a first step to a more involved and more useful extension.

7.5 A Minimal Cost C-cover.

One of the more obvious deficiencies of the solution to the targeting problem of section one is that when only one 1-set is computed, that particular set might include a target very heavily fortified whereas one not as heavily fortified might have been a member of another minimal α-set. One approach to remedy this deficiency would be to compute all minimal α-sets; and indeed the approach will be mentioned subsequently. However, it is also possible that for some variety of reasons, it would be preferable even to destroy more than the minimum number of targets. The term, preferable, indicates that there might be a utility function or cost function associated with the problem.

The extension of the algorithm so that it may handle costs associated with the columns is perhaps the most interesting extension that we shall discuss. The author believes that this extension might result in a decrease in the computation time required. The belief is based upon the observation that the lower bounds calculated in the present program are relatively close to each other. Thus there is entirely too much switching away from one branch, to another, and then back to the original branch. With a wide difference among the column costs, however, the differences among the various estimates of the C-width of the original matrix should be equally wide. This will serve to reduce the unnecessary switching from branch to branch. That is, it is more likely that when a branch is dropped by the algorithm, it is because that branch has become unprofitable.

Suppose we assign to each column of the matrix a cost, which need not be integral, and may be positive, negative, or zero. We would then be interested in finding a C-cover (or an α-set) which has

minimum cost associated with it. Of course, such a C-cover might not be a minimal C-cover as defined previously.

The modification to the computer program would be surprisingly simple. The cost vector would be read in, and let us assume stored in COMMON. Now the "infinity" for unfeasible column combinations must be increased to some arbitrarily large number. An estimate of $\varepsilon(C)$ would be calculated for each sub-matrix using the same subroutine as presently used for $\alpha > 1$. From subroutine BOUND, however, the program would enter a new subroutine, such as subroutine COST presented in Figure 27. In this subroutine, a cost for the ρ just computed would be estimated. The estimate would be optomistic in the sense that the cost for the columns would be the sum of the smallest cost components not already used on this branch. For example, consider a cost vector, (1,2,3). Assume that, on the current branch, column one has been either included or excluded, and that we have computed $\rho = 1$. Then the cost for the sub-matrix would be estimated as two; and the estimate of the cost for the minimal C-cover would be two plus the cost of column one, if column one had been included, or two, if column one had been excluded.

Finally, either in the same subroutine, or in the main program, the cost of the set of currently included columns would be computed and stored in place of the argument, VCOL(I) of the current program. Also, VEPSILON(I) of the current program would be replaced by the sum of VCOL(I) and the cost estimate just computed in subroutine COST, as described in the above paragraph.

The reader is reminded that the subroutine COST of Figure 27 has not been checked out and that the remarks in this section about decreasing total computation time represent merely the author's intuition and are not based upon observations.

```
      SUBROUTINE COST (EPSILON, I)
      COMMON/BLOCKH/CCOST(144)/BLOCKB/IDATA(1761)
      EQUIVALENCE (MASKC,IMASK(3))
      TYPE LOGIC5 (3) MASKC, BIT
      DIMENSIONS TEMP(144)
      DO 5 j = 1, 3
    5 IMASK(J) = IDATA((587*(I-1))+582+J)
      N = IDATA((587*(I-1))+3)
      BIT = BIT * MASKC
      IF (BIT.EQ.0) 10, 20
   10 K = K + 1
      TEMP(K) = CCOST(J)
      IF (TEMP(K).LT.TEST) 15, 20
   15 TEST = TEMP(K)
      K1 = K
      L = EPSILON
   20 CONTINUE
      IF (K.LT.L) 25, 30
   25 EPSILON = 1.E+20
      RETURN
   30 EPSILON = TEST
      J = 1
   35 J = J + 1
      TEST1 = TEMP(1)
      DO 45 M = 2, K
      IF (TEMP(K).GE.TEST.AND.TEMP(M).LT.TEST1.AND.K1.NE.M) 40, 45
   40 TEST1 = TEMP(M)
   45 CONTINUE
      EPSILON = EPSILON + TEST1
      TEST = TEST1
      IF (J.EQ.L) 50, 35
   50 RETURN
      END
```

Figure 27

Once both of the above extensions have been programmed we may use the algorithm to solve a large variety of problems which are a sub-class of the set of integer programming problems.

7.6 An Integer Program with a (0,1) Constraint Matrix.

We merely point out in this section, a formulation of a problem which the extended algorithm can solve. Given the system of linear inequalities:

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

where a_{ij} is either zero or one; find values for each x_j such that x_j is either zero or one, which minimizes:

$$\sum_{j=1}^{n} c_j \cdot x_j .$$

Here the c_j are costs, and the b_i are the components of what has previously been called the C vector.

7.7 Constraints on Combinations of Columns.

Suppose that upon any of the problems which may be solved by extensions of the algorithm, we impose constraints of the following type: if column a is included in the C-cover, then column b must be excluded.

We could write a relatively short subroutine to handle this type of constraint. It would be necessary to put the constraints in a convenient form, say for each constraint construct a mask of zeros except in the bit positions corresponding to columns which cannot be included together. For example, let there be five columns and assume two constraints: that columns one and two cannot be included together, and that columns three and five cannot be included together.Then the two masks would be:

11000 and 00101.

We put these constraints into the program in some convenient fashion (probably by the same system used for putting the A matrix in the current program); and write a subroutine to compute the logical product of each constraint with the mask of included columns. If there are no ones in the product for any constraint, the subroutine must set a "current" cost vector equal to the input cost vector. If there are two or more ones in any single product, the subroutine must indicate that an infeasible column selection has been made. Finally, for each constraint with exactly one "1" in the product, set the "current" cost of every "1" in the constraint to "infinity"; except of course, the "1" representing the current column inclusion. We use the "current" cost vector in computing bounds instead of the input cost vector.

7.8 Finding All Minimal C-covers of the Matrix.

It is possible to simplify the search for minimal C-covers once the first one has been located. The same algorithm applies except that we have information to rule out as infeasible, any set of columns which yields an estimate of $\varepsilon(C)$ larger than the computed C-width. Although the simplification would contribute to a substantial savings in computation time for each additional minimal C-cover, it is believed that for most problems the search for all minimal C-covers would require more computation time than the results would warrant.

BIBLIOGRAPHY

1. Berge, Claude. The Theory of Graphs and Its Applications. John Wiley & Sons, 1958.

2. Ford, L. R., Jr., and Fulkerson, D. R. Flows in Networks. Princeton University Press, 1962.

3. Fulkerson, D. R., and Ryser, H. J. "Multiplicities and Minimal Widths for (0,1) Matrices." Canadian Journal of Mathematics, Vol. 14, 1962, pp. 498-508.

4. Fulkerson, D. R., and Ryser, H. J. "Widths and Heights of (0,1) Matrices." Canadian Journal of Mathematics, Vol. 13, 1961, pp. 239-255.

5. Fulkerson, D. R., and Ryser, H. J. "Width Sequences for Special Classes of (0,1) Matrices". Canadian Journal of Mathematics. Vol. 15, 1963, pp. 371-396.

6. Harrison, Gordon A. United States Army in World War II, European Theater of Operations; Cross Channel Attack, Vol. I. Government Printing Office, 1951.

7. Little, John D. C.; Murty, Katta G.; Sweeney, Dura W.; and Karel, Caroline. "An Algorithm for the Traveling Salesman Problem" Operations Research, Vol. 11, 1963, pp. 972-989.

8. Ray-Chaudhuri, D. K. "An Algorithm for a Minimum Cover of an Abstract Complex". Canadian Journal of Mathematics, Vol. 15, 1963, pp. 11-24.

9. Ryser, H. J. "Combinatorial Properites of Matrices of Zeros and Ones". Canadian Journal of Mathematics, Vol. 9, 1957, pp. 371-377.

10. Ryser, H. J. "Matrices of Zeros and Ones". Bulletin of the American Mathematical Society, Vol. 66, 1960, pp. 442-464.

APPENDIX I

Listing of PROGRAM WIDTH

```
      PROGRAM WIDTH
C
      TYPE INTEGER A5, ALPHA, COL1, COL2, COL3, COLSET, EPSILON,
     1   EPSILON1, EPSILON2, EPSILON3, R, R1, R2, R3, RO1, RO2, RO3, S1,
     2   S2, S3, SO1, SO2, SO3, T, VEPSILON,SUM, RT, VCOL
      TYPE LOGIC5 (3) A, BIT, COLUMNS, MASK1, MASK2, MASK3, MASKC1,
     1   MASKC2, MASKC3, ROW, MASKR, VMASK, VMASKC
C
      COMMON/BLOCKA/A(144)/BLOCKB/IDATA(1761)/BLOCKC/MO,NO,ALPHA,EPSILON
     1   ,RT(144)/BLOCKD/VEPSILON(2000)/BLOCKE/ROW,SUM/BLOCKG/IRAND
      DIMENSION A5(3,144), COLSET(144), R1(144),R2(144),
     1   R3(144), RO1(144),RO2(144), RO3(144), S1(144), S2(144),
     2   S3(144), SO1(144), SO2(144), SO3(144), IDMP(587) , ITEMP(2000),
     3   VMASK(2000), VMASKC(2000), VCOL(2000)
C
      EQUIVALENCE (A5,A), (IDATA(1),I1), (IDATA(2),M1), (IDATA(3),N1),
     1   (IDATA(4),R1), (IDATA(148),RO1), (IDATA(292),S1), (IDATA(436),
     2   SO1), (IDATA(580),MASK1), (IDATA(583),MASKC1), (IDATA(586),COL1
     3   ),(IDATA(587),EPSILON1), (IDATA(588),I2),(IDATA(589),M2),
     4   (IDATA(590),N2),(IDATA(591),R2),(IDATA(735),RO2),(IDATA(879)
     5   ,S2),(IDATA(1023),SO2), (IDATA(1167),MASK2),(IDATA(1170),
     6   MASKC2), (IDATA(1173),COL2), (IDATA(1174),EPSILON2),
     7   (IDATA(1175),I3), (IDATA(1176),M3), (IDATA(1177),N3), (IDATA
     8   (1178),R3), (IDATA(1322),RO3),(IDATA(1466),S3), (IDATA(1610),
     9   SO3), (IDATA(1754),MASK3), (IDATA(1757),MASKC3), (IDATA(1760),
     *   COL3), (IDATA(1761),EPSILON3)
C
C    READ IN INITIAL OCTAL RANDOM NUMBER.  SET ALL DATA WORDS TO ZERO
C
      READ (4,900) IRAND
    5 DO 10 I = 1,1761
   10 IDATA(I) = 0
      DO 15 I = 1, 144  $        RT(I) = 0
   15 COLSET(I)=0
      DO 20 I = 1, 500  $  VMASK(I) = VMASKC(I) = VCOL(I) = 0
   20 VEPSILON(I) = 0
```

```
C                                                                       WID00037
C    INPUT DIMENSIONS OF MATRIX,(MAXIMUM 144 BY 144), ALPHA, AND IF      WID00038
C    MATRIX IS TO BE RANDOMLY GENERATED, THE MAXIMUM NUMBER OF ONES      WID00039
C    IN A ROW.  THIS NUMBER MUST BE GREATER THAN ALPHA.  IF ALPHA IS     WID00040
C    NEGATIVE MATRIX WILL BE GENERATED RANDOMLY.  IF POSITIVE, MATRIX    WID00041
C    WILL BE READ IN FROM PUNCHED CARDS.  IF ZERO, COMPUTER RUN          WID00042
C    TERMINATES.                                                         WID00043
C                                                                       WID00044
C    PREPARE MATRIX FOR INPUT ON PUNCHED CARDS BY CONVERTING ZFROS       WID00045
C    AND ONES IN EACH ROW TO OCTAL DIGITS IN GROUPS OF THREE STARTING    WID00046
C    FROM THE LEFT.  PUNCH ONE CARD FOR EACH ROW, COLUMNS 1 THROUGH 48.  WID00047
C    EXAMPLE--M = 15, A ROW OF THE MATRIX IS 000011101111110.  PUNCH     WID00048
C    THE CARD STARTING IN COLUMN ONE AS--03576.                          WID00049
C                                                                       WID00050
   25 READ (4,1000) MO,NO,ALPHA,NONES                                    WID00051
      IF(ALPHA) 30, 40,45                                                WID00052
   30 ALPHA = -ALPHA                                                     WID00053
   35 CALL RANDOM(MO,NO,ALPHA,NONES)                                     WID00054
      GO TO 50                                                           WID00055
C                                                                       WID00056
   40 PRINT 1100                                                         WID00057
      STOP                                                               WID00058
C                                                                       WID00059
   45 READ (4,1200)((A5(I,J),I=1,3),J=1,MO)                              WID00060
C                                                                       WID00061
C   ALL WORDS RESERVED FOR THE MATRIX BUT UNUSED IN THIS PROBLEM         WID00062
C    ARE SET TO ZERO.  MATRIX IS PRINTED OUT.                            WID00063
C                                                                       WID00064
   50 IF(NO.GT.48) 65, 55                                                WID00065
   55 DO 60 I=1,MO                                                       WID00066
   60 A5(2,I) = A5(3,I) = 0  $  GO TO 80                                 WID00067
   65 IF(NO.GT.96) 80,70                                                 WID00068
   70 DO 75 I=1,MO                                                       WID00069
   75 A5(3,I)=0                                                          WID00070
   80 IF(MO.LT.144) 85,95                                                WID00071
   85 I= MO+1  $  DO 90 J=I,144                                          WID00072
```

```
   90 A5(1,J) = A5(2,J) =A5(3,J) = 0
C
   95 PRINT 1300, MO,NO,((A5(I,J),I=1,3),J=1,MO)
C
C    START COMPUTATION OF ALPHA-WIDTH
C
      CALL TIME (ITIME1)  $  I4 = IO = I1 = 1  $  JO = 1
  100 DO 105 I = 1, MO
  105 RO1(I) = I
  110 CALL ROWSUM(1)
      GO TO (115,120,120,120) I4
C
C    CHECK FOR PROBLEM FEASIBILITY
C    LOCATE ALL COLUMNS WHICH MUST BE IN EVERY ALPHA-SET
C
  115 DO 1154 I = 1, MO
      IF (R1(RO1(I)) - ALPHA) 140,1151,1154
 1151 DO 1153 J = 1, NO
      BIT = BIT/J
      IF (BIT*A(RO1(I)).EQ.0) 1153,1152
 1152 MASK1 = MASK1 + BIT  $  MASKC1 = MASKC1 + BIT
 1153 CONTINUE
 1154 CONTINUE  $  I4 = 2  $  ROW = MASK1  $  CALL SHIFTSUM
      VCOL(1) = SUM  $  GO TO 110
C
C    COMPUTE RT OF THE BRANCHING MATRIX.
C
  120 DO 1201 I = 1,MO
      ROW = MASK1*A(I)
      CALL SHIFTSUM  $  IDIFF = ALPHA - SUM
      RT(I) = XMAXOF(IDIFF,0)
 1201 CONTINUE  $  GO TO (125,125,125,160) I4
C
C    MASK OUT ROWS SUFFICIENTLY REPRESENTED
C
  125 I = 0
```

```
 1251 I = I + 1  $  IF (R1(RO1(I)).GT.0.AND.RT(RO1(I)).LE.0) 1252,1254
 1252 R1(RO1(I)) = 0  $  M1 = M1 - 1  $  DO 1253 J = 1, M1
 1253 RO1(J) = RO1(J+1)  $  RO1(M1+1) = 0  $  I = I - 1
 1254 IF (I.LT.M1) 1251,1255
C
C   TEST FOR TERMINATION.  IF TEST FAILS COMPUTE REMAINDER OF
C   MATRIX PARAMETERS
C
 1255 IF (M1.EQ.0) 130,135
  130 COLUMNS = MASK1  $  GO TO 240
  135 CALL BIGSORT(1)
      CALL COLSUM(1)
      CALL BIGSORT(2)
      GO TO (155,155,160,160) I4
  140 PRINT 1400, ALPHA  $  GO TO 5
  155 VMASK(1) = MASK1  $  VMASKC(1) = MASKC1
      EPSILON1 = VEPSILON(1) = 2*NO
C
C
C   PARAMETERS ENDING IN 1 REFER TO BRANCHING MATRIX.  THOSE ENDING
C   IN 2 REFER TO SUB-MATRIX WITH COLUMN INCLUDED IN ALPHA-SET
C   THOSE ENDING IN 3 REFER TO SUB-MATRIX WITH COLUMN EXCLUDED
C   FROM ALPHA-SET.
C
  160 DO 161 I = 588, 1761
  161 IDATA(I) = 0  $  IO = I2 = IO + 1  $  JO = JO + 1
C
C   TEST FOR COMPLETION
C
      IF (N1.EQ.0) 166,167
  166 COLUMNS = MASK1  $  GO TO 240
C
C   DERIVE SUB-MATRIX WITH BRANCHING COLUMN INCLUDED IN ALPHA-SET
C
  167 BIT = BIT / SO1(1)
      VMASK(JO) = MASK2 = MASK1 + BIT
```

```
      VMASKC(JO) = MASKC2 = MASKC1 + BIT                                WID00145
      CALL ROWSUM(2)                                                    WID00146
      IF(M2.EQ.0) 169,174                                               WID00147
  169 IF (EPSILON1+1.LF.MIN(JO-1)) 170,172                              WID00148
  170 COLUMNS = MASK2  $  GO TO 240                                     WID00149
  172 EPSILON2=VEPSILON(JO)=EPSILON1+1                                  WID00150
      VCOL(JO) = COL2 = COL1 + 1  $  GO TO 192                          WID00151
C                                                                       WID00152
  174 I = 0                                                             WID00153
  175 I = I + 1                                                         WID00154
      ROW = MASK2*A(RO2(I))                                             WID00155
      CALL SHIFTSUM  $  IF (SUM.GE.ALPHA) 180, 190                      WID00156
  180 R2(RO2(I)) = 0  $  M2 = M2 - 1  $  IF (M2.EQ.0) 169, 184          WID00157
  184 DO 185 J = I, M2                                                  WID00158
  185 RO2(J) = RO2(J+1)  $  RO2(M2+1) = 0   $  I = I - 1                WID00159
  190 IF (I.LT.M2) 175, 191                                             WID00160
  191 CONTINUE                                                          WID00161
      CALL BIGSORT(3)                                                   WID00162
      CALL COLSUM(2)                                                    WID00163
      CALL BIGSORT(4)                                                   WID00164
      CALL BOUND(2)                                                     WID00165
C                                                                       WID00166
C   IF ALLOWED MEMORY IS USED UP, ATTEMPT TO DISCARD DATA NO            WID00167
C   LONGER NEEDED.  IF UNSUCCESSFUL, TERMINATE.                         WID00168
C                                                                       WID00169
  192 IF (JO.GE.2000) 1920, 193                                         WID00170
 1920 N8 = 2 * NO  $  JO = 0                                            WID00171
      DO 1922 I = 1, 2000                                               WID00172
      IF (VEPSILON(I) .LT.N8) 1921, 1922                                WID00173
 1921 JO = JO + 1  $  ITEMP(JO) = I                                     WID00174
 1922 CONTINUE  $  IF (JO.EQ.2000) 1923, 1924                           WID00175
 1923 PRINT 1700, IO  $  GO TO 5                                        WID00176
 1924 DO 1925 I = 1, JO  $  K = ITEMP(I)                                WID00177
      VEPSILON(I) = VEPSILON(K)  $  VMASK(I) = VMASK(K)                 WID00178
      VMASKC(I) = VMASKC(K)                                             WID00179
 1925 VCOL(I) = VCOL(K)  $  J1 = 2*((JO+1)/2)                           WID00180
```

```
      IF (J1.GT.JO) 1926, 193
 1926 VEPSILON(J1) = N8  $  JO = J1
  193 VCOL(JO) = COL2 = COL1 + 1  $  VEPSILON(JO)=EPSILON2=EPSILON+COL2
C
C   DERIVE SUB-MATRIX WITH BRANCHING COLUMN EXCLUDED FROM ALPHA-SET
C
  194 IO = I3 = IO + 1
      JO = JO + 1
      VMASK(JO) = MASK3 = MASK1
      VMASKC(JO) = MASKC3 = MASKC2
      VCOL(JO) = COL3 = COL1
      CALL ROWSUM(3)
      DO 200 I = 1,MO
      IF (R3(I) .LT. RT(I)) 195,200
  195 EPSILON3 = VEPSILON(JO) = 2*NO  $  GO TO 205
  200 CONTINUE
      CALL BIGSORT(5)
      CALL COLSUM(3)
      CALL BIGSORT(6)
      CALL BOUND(3)
      EPSILON3 =VEPSILON(JO) = EPSILON + COL3
C
C   CHOOSE NEXT BRANCHING MATRIX
C
  205 T = MIN(JO)  $  IF (T-JO+1) 210, 225, 230
  210 DO 215 I = 1, 587
  215 IDATA(I) = 0   $  I4 = 3
      MASK1 = VMASK(T)  $  MASKC1 = VMASKC(T)  $  COL1 = VCOL(T)$I1=T
      EPSILON1 = VEPSILON(T)  $  VEPSILON(T) = 2*NO  $  GO TO 100
C
  225 I = 2  $  GO TO 233
  230 I = 3
C
  233 DO 235 J = 1,587
  235 IDATA(J) = IDATA(J + 587*(I-1))  $  VEPSILON(T) = NO*2
      I4 = 4  $  GO TO 120
```

```
C
C   . PREPARE OUTPUT
C
  240 J = 0  $  DO 250 I = 1,NO  $  BIT = BIT/I
      IF(BIT*COLUMNS.EQ.0) 250,245
  245 J = J + 1  $  COLSET(J) = I
  250 CONTINUE
C
      CALL TIME(ITIME2)
      ITIME = ITIME2 - ITIME1  $  ITIME1 = ITIME/60
      ITIME2 = ITIME1/60
      ITIME = ITIME1 - ITIME2*60
C
      PRINT 1500, ALPHA,J,IO,ITIME2, ITIME
      PRINT 1600, (COLSET(K),K=1,J)
      GO TO 5
C
  900 FORMAT (O16)
 1000 FORMAT (2I3,I4,I3)
 1100 FORMAT(13H END OF DATA.)
 1200 FORMAT(3O16)
 1300 FORMAT(1H1, 41X, 23HA MATRIX OF DIMENSION (,I3, 6H) BY (, I3,
     12H).///////(36X,3O16/))
 1400 FORMAT(///39X,I3,38H-WIDTH DOES NOT EXIST FOR THIS MATRIX.)
 1500 FORMAT(1H0,44X,I3,24H-WIDTH OF THE MATRIX IS ,I3,1H.,////
     131H NUMBER OF ITERATIONS REQUIRED-,I5,/23H TIME FOR COMPUTATION-
     2I2,4HMIN,I2,4HSEC.)
 1600 FORMAT(// 43H0COLUMNS IN MINIMAL REPRESENTATIVE SET ARE-,
     1//(15X,14(I3,1H,2X)//)
 1700 FORMAT (28H0ALLOWED MEMORY EXCEEDED AT ,I5, 12H ITERATIONS.)
C
      END
```

```
      FUNCTION MIN(K)                                                   MIN00001
C                                                                       MIN00002
C    LOCATE THE SMALLEST COMPONENT OF A VARIABLE SIZE VECTOR            MIN00003
C    AND RETURN ITS SUBSCRIPT                                           MIN00004
C                                                                       MIN00005
      COMMON/BLOCKD/IVEC(2000)                                          MIN00006
      MIN = 1  $  ITEST = IVEC(1)                                       MIN00007
      DO 10 J = 2,K                                                     MIN00008
      IF(ITEST.LE.IVEC(J)) 10,5                                         MIN00009
    5 ITEST = IVEC(J)  $  MIN = J                                       MIN00010
   10 CONTINUE                                                          MIN00011
      RETURN                                                            MIN00012
      END                                                               MIN00013
```

```
      SUBROUTINE ROWSUM(I)
C
C    COMPUTE THE LOGICAL PRODUCT OF A ROW OF THE MATRIX AND A MASK, AND
C    COMPUTE THE SUM OF THE RESULT
C
      COMMON/BLOCKA/A(144)/BLOCKB/IDATA(1761)/BLOCKC/M,N,DUMMY(146)/
     1   BLOCKE/ROW,SUM
      TYPE INTEGER SUM
      TYPE LOGIC5 (3) A, MASK, MASK1, ROW
      DIMENSION MASKI(3)
      EQUIVALENCE (MASKI,MASK)
C
      GO TO (5,10,15) I
    5 DO 6 K = 1, 3
    6 MASKI(K) = IDATA(582+K)  $  J = 3  $  L1 = 2  $  GO TO 20
   10 DO 11 K = 1, 3
   11 MASKI(K) = IDATA(1169+K)  $  J = 590  $  L1 = 589  $  GO TO 20
   15 DO 16 K = 1, 3
   16 MASKI(K) = IDATA(1756+K)  $  J = 1177  $  L1 = 1176
   20 MASK1 = -MASK  $  JO = J + 144
      DO 30 K = 1, M
      GO TO (22,21,21) I
   21 IF (IDATA(3+K).EQ.0) 30, 22
   22 ROW = MASK1*A(K)
      CALL SHIFTSUM  $  IF (SUM.EQ.0) 30, 25
   25 IDATA(L1) = IDATA(L1) + 1
      IDATA(J+K) = SUM
      IDATA(JO + IDATA(L1)) = K
   30 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE COLSUM (K)
C
C     COMPUTE THE SUM OF EACH COLUMN OF THE MATRIX APPROPRIATELY MASKED
C
      COMMON/BLOCKA/A(144)/BLOCKB/IDATA(1761)/BLOCKC/M,N,DUMMY(146)
      TYPE LOGIC5 (3) A, BIT, MASK
      DIMENSION MASKI(3)
      EQUIVALENCE (MASKI,MASK)
      GO TO (5,10,15) K
    5 DO 6 I=1,3
    6 MASKI(I)=IDATA(582+I) $ L=291 $ J=435 $ KO=3 $ GO TO 20
   10 DO 11 I=1,3
   11 MASKI(I)=IDATA(1169+I) $ L=878 $ J=1022 $ KO=590 $ GO TO 20
   15 DO 16 I = 1,3
   16 MASKI(I)=IDATA(1756+I) $ L=1465 $ J=1609 $ KO=1177
   20 IDATA(KO) = 0
      DO 45 I = 1, N  $  BIT = BIT/I  $  IDATA(L+I) = 0
      IF(BIT + MASK.EQ.MASK) 45,25
   25 DO 35 IO = 1,M
      IF (IDATA(KO+IO).EQ.0.OR.BIT*A(IO).EQ.0) 35,30
   30 IDATA(L+I) = IDATA(L+I) + 1
   35 CONTINUE
      IF(IDATA(L+I).EQ.0) 45,40
   40 IDATA(KO) = IDATA(KO) + 1 $  IDATA(J+IDATA(KO)) = I
   45 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE BIGSORT (I)                                            SRT00001
C                                                                       SRT00002
C     GIVEN AN INPUT VECTOR, GENERATE A VECTOR OF SUBSCRIPTS SO THAT THE SRT00003
C     GENERATED VECTOR ORDERS THE INPUT VECTOR IN INCREASING ORDER      SRT00004
C                                                                       SRT00005
      COMMON/BLOCKB/IDATA(1761)                                         SRT00006
C                                                                       SRT00007
      GO TO (5,10,15,20,25,30) I                                        SRT00008
    5 J = 2  $  K = 3  $  L = 147  $  GO TO 35                          SRT00009
   10 J = 3  $  K = 291  $  L = 435  $  GO TO 35                        SRT00010
   15 J = 589  $  K = 590  $  L = 734  $  GO TO 35                      SRT00011
   20 J = 590  $  K = 878  $ L = 1022  $  GO TO 35                      SRT00012
   25 J = 1176  $  K = 1177  $  L = 1321  $  GO TO 35                   SRT00013
   30 J = 1177  $  K = 1465  $  L = 1609                                SRT00014
C                                                                       SRT00015
   35 M = IDATA(J)  $  DO 45 N = 1,M  $  ITEST  = IDATA(K + IDATA(L+N)) SRT00016
      NO = N + 1  $  DO 45 N1 = NO,M                                    SRT00017
      IF (ITEST.GE.IDATA(K+IDATA(L+N1)))  45,40                         SRT00018
   40 ITEST = IDATA(K + IDATA(L + N1))                                  SRT00019
      ITEMP = IDATA( L+N)  $ IDATA(L+N) = IDATA(L+N1)                   SRT00020
      IDATA(L + N1) = ITEMP                                             SRT00021
   45 CONTINUE                                                          SRT00022
      RETURN                                                            SRT00023
      END                                                               SRT00024
```

```
      SUBROUTINE BOUND (I)                                              BND00001
C                                                                       BND00002
C    COMPUTE A LOWER BOUND ON THE ALPHA-WIDTH OF A CLASS OF MATRICES    BND00003
C     IF ALPHA EQUALS ONE, A GREATEST LOWER BOUND IS COMPUTED           BND00004
C                                                                       BND00005
      COMMON/BLOCKB/IDATA(1761)/BLOCKC/I1,I2,ALPHA,EPSILON,RT(144)      BND00006
      TYPE INTEGER ALPHA, EPSILON,  E, F, QO, QF,  RT                   BND00007
      GO TO (5,10,15) I                                                 BND00008
    5 M = IDATA(2) $ N = IDATA(3) $ K = 3 $ GO TO 20                    BND00009
   10 M= IDATA(589) $ N = IDATA(590) $ K = 590 $ GO TO 20               BND00010
   15 M = IDATA(1176) $ N = IDATA(1177) $ K = 1177                      BND00011
C                                                                       BND00012
   20 KO = K + 144  $  KS = KO + 144  $  K1 = KS + 144                  BND00013
C                                                                       BND00014
      IF (IDATA(KS+IDATA(K1+1)).EQ.0) 25,30                             BND00015
   25 PRINT 1000, I  $  STOP 100                                        BND00016
C                                                                       BND00017
   30 IF (ALPHA.GT.1) 85,31                                             BND00018
   31 EPSILON = ALPHA*M/IDATA(KS + IDATA(K1+1))  $  F = EPSILON  $ E = 0BND00019
      QO = (-ALPHA)*M                                                   BND00020
      DO 35 J = 1,M                                                     BND00021
   35 QO = QO + IDATA(K+IDATA(KO+J))                                    BND00022
   40 QF = QO                                                           BND00023
   45 F = F + 1                                                         BND00024
      IF(F.GT.N) 65,50                                                  BND00025
   50 QF = QF + E - IDATA(KS+IDATA(K1+F))                               BND00026
   55 IF(QF.LT.0) 60,45                                                 BND00027
   60 EPSILON = EPSILON + 1 $  F = EPSILON  $  E = 0                    BND00028
      IF (EPSILON.GT.N) 80,61                                           BND00029
   61 QO = QO + IDATA(KS + IDATA(K1 + EPSILON))  $  GO TO 40            BND00030
   65 QF = QO  $  F = EPSILON  $  E = E + 1  $  IF (E.GT.M) 75,66       BND00031
   66 QF = QF + ALPHA - IDATA(K + IDATA(KO + E))  $  IF(QF.LT.0) 60,70  BND00032
   70 IF(E.GT.M) 75,55                                                  BND00033
   75 RETURN                                                            BND00034
   80 EPSILON = I2*2  $  RETURN                                         BND00035
C                                                                       BND00036
```

```
   85 IBND = 0  $  DO 90 J = 1,M                                   BND00037
   90 IBND = IBND + RT(IDATA(KO+J)) . $  ITEST = 0                 BND00038
      DO 95 J = 1,N                                                BND00039
      ITEST = ITEST + IDATA(KS+IDATA(K1+J))                        BND00040
      IF (ITEST.GE.IBND) 100,95                                    BND00041
   95 CONTINUE  $  EPSILON = 2*I2  $  RETURN                       BND00042
  100 EPSILON = J  $  RETURN                                       BND00043
C                                                                  BND00044
 1000 FORMAT(48H1ALGORITHM HAS PROCEEDED PAST TERMINATION POINT.10X,I1) BND00045
      END                                                          BND00046
```

```
      SUBROUTINE RANDOM (MO,NO,ALPHA,NONES)
C
C    GENERATE A MATRIX WHOSE ROWS ARE RANDOMPLY GENERATED ACCORDING
C    TO A UNIFORM DISTRIBUTION BETWEEN ALPHA AND NONES
C
      COMMON/BLOCKA/A(144)/BLOCKE/ROW,SUM/BLOCKF/WORD
      TYPE LOGIC5 (3) A, BIT, MASK, ROW, WORD
      TYPE INTEGER ALPHA, SUM
      MASK = 0
      DO 10 I = 1, NO
      BIT = BIT/I
   10 MASK = MASK + BIT
      I = 0
   15 I = I + 1
      ROW = MASK
   16 CALL RANDGEN
      ROW = ROW * WORD
      IF (NONES.EQ.0) 19, 17
   17 CALL SHIFTSUM
      IF (SUM.GT.NONES) 16, 19
   19 IF (SUM.LT.ALPHA) 20, 25
   20 I = I - 1
      GO TO 30
   25 A(I) = ROW
   30 IF (I.LT.MO) 15, 35
   35 RETURN
      END
```

```
         IDENT        RANDGEN                                                    GEN00001
BLOCKF   BLOCK        3                                                          GEN00002
         COMMON       WORD(3)                                                    GEN00003
BLOCKG   BLOCK        1                                                          GEN00004
         COMMON       R                                                          GEN00005
         ENTRY        RANDGEN                                                    GEN00006
RANDGEN  SLJ          **                THREE RANDOM NUMBERS ARE                 GEN00007
         SIU          ENDING               GENERATED BY THE FORMULA              GEN00008
         ENI      1   2                    X(I+1)=X(I)*2**10+X(I)+101            GEN00009
         ENQ          0                                                          GEN00010
A1       LDA          R                                                          GEN00011
         LLS          10                                                         GEN00012
         ADD          R                                                          GEN00013
         INA          101                                                        GEN00014
         STA          R                                                          GEN00015
         STA      1   WORD                                                       GEN00016
         IJP      1   A1                                                         GEN00017
ENDING   ENI      1   **                                                         GEN00018
         SLJ          RANDGEN                                                    GEN00019
         END                                                                     GEN00020
```

```
          IDENT      TIME                                                TIM00001
          ENTRY      TIME                                                TIM00002
TIME      SLJ        **                READ THE CURRENT VALUE OF THE     TIM00003
          LDA        *                    REAL TIME CLOCK                TIM00004
          ALS        24                                                  TIM00005
          SAU        A1                                                  TIM00006
          INA        1                                                   TIM00007
          SAU        TIME                                                TIM00008
A1        LDA        24                                                  TIM00009
          ALS        24                                                  TIM00010
          SAU        A2                                                  TIM00011
          ENQ        0                                                   TIM00012
          LDA     7  =00                                                 TIM00013
A2        STA        **                                                  TIM00014
          SLJ        TIME                                                TIM00015
          END                                                            TIM00016
```

```
         IDENT      SHIFTSUM                                      SUM00001
         ENTRY      SHIFTSUM                                      SUM00002
BLOCKE   BLOCK      4                                             SUM00003
         COMMON     ROW(3),SUM                                    SUM00004
SHIFTSUM SLJ        **                THIS ROUTINE COMPUTES       SUM00005
         SIU     1  ENDING              THE NUMBER OF 1-BITS IN   SUM00006
         SIL     2  ENDING              UP TO 3 CONSECUTIVE LOCATIONSSUM00007
         ENA        0                   OF MEMORY.                SUM00008
         STA        SUM                                           SUM00009
         FNI     1  2                                             SUM00010
A1       LDQ     1  ROW                                           SUM00011
         ENI     2  47                                            SUM00012
A2       QJP     3  *+1                                           SUM00013
         SLJ        *+2                                           SUM00014
+        RAO        SUM                                           SUM00015
+        IJP     2  *+1                                           SUM00016
         SLJ        *+2                                           SUM00017
+        QLS        1                                             SUM00018
         SLJ        A2                                            SUM00019
+        IJP     1  A1                                            SUM00020
ENDING   ENI     1  **                                            SUM00021
         FNI     2  **                                            SUM00022
         SLJ        SHIFTSUM                                      SUM00023
         END                                                      SUM00024
```

```
         IDENT    Q1QMATH                                          MTH00001
         REM               THIS ROUTINE PERFORMS ALL TYPE          MTH00002
ACC      OCT      0,0,0      LOGIC5 MATH OPERATIONS.               MTH00003
         ENTRY    Q1Q00500                                         MTH00004
Q1Q00500 SLJ      **       LOAD EACH WORD OF THE                   MTH00005
         LDA      *          ACCUMULATOR WITH THE SAME             MTH00006
         RTJ      INITIAL    INTEGER CONSTANT.                     MTH00007
         NOP                                                       MTH00008
+        SAU      *+1                                              MTH00009
+        LDA      **                                               MTH00010
         STA      ACC+2                                            MTH00011
         ENA      0                                                MTH00012
         STA      ACC                                              MTH00013
         STA      ACC+1                                            MTH00014
         SLJ      Q1Q00500                                         MTH00015

         ENTRY    Q1Q00550                                         MTH00016
Q1Q00550 SLJ      **       LOAD THE ACCUMULATOR WITH               MTH00017
         LDA      *          A LOGIC5 CONSTANT.                    MTH00018
         RTJ      INITIAL                                          MTH00019
         NOP                                                       MTH00020
+        SAU      A1                                               MTH00021
         INA      1                                                MTH00022
         SAU      A2                                               MTH00023
         INA      1                                                MTH00024
         SAU      A3                                               MTH00025
A1       LDA      **                                               MTH00026
         STA      ACC                                              MTH00027
A2       LDA      **                                               MTH00028
         STA      ACC+1                                            MTH00029
A3       LDA      **                                               MTH00030
         STA      ACC+2                                            MTH00031
         SLJ      Q1Q00550                                         MTH00032
```

```
         ENTRY    Q1Q01550                                      MTH00033
Q1Q01550 SLJ      **                LOAD THE ACCUMULATOR WITH   MTH00034
         LDA      *                   THE COMPLEMENT OF A       MTH00035
         RTJ      INITIAL             LOGIC5 CONSTANT.          MTH00036
         NOP                          (UNARY -OPERATOR)         MTH00037
+        SAU      B1                                            MTH00038
         INA      1                                             MTH00039
         SAU      B2                                            MTH00040
         INA      1                                             MTH00041
         SAU      B3                                            MTH00042
B1       LAC      **                                            MTH00043
         STA      ACC                                           MTH00044
B2       LAC      **                                            MTH00045
         STA      ACC+1                                         MTH00046
B3       LAC      **                                            MTH00047
         STA      ACC+2                                         MTH00048
         SLJ      Q1Q01550                                      MTH00049
```

```
         ENTRY    Q1Q02550                                          MTH00050
Q1Q02550 SLJ      **                 FORM THE LOGICAL SUM OF        MTH00051
         LDA      *                    THE ACCUMULATOR AND A        MTH00052
         RTJ      INITIAL                LOGIC5 CONSTANT.           MTH00053
         NOP                              (+OPERATOR)               MTH00054
+        SAL      C1                                                MTH00055
         INA      1                                                 MTH00056
         SAU      C2                                                MTH00057
         INA      1                                                 MTH00058
         SAL      C3                                                MTH00059
C1       LDA      ACC                                               MTH00060
         SST      **                                                MTH00061
         STA      ACC                                               MTH00062
         LDA      ACC+1                                             MTH00063
C2       SST      **                                                MTH00064
         STA      ACC+1                                             MTH00065
C3       LDA      ACC+2                                             MTH00066
         SST      **                                                MTH00067
         STA      ACC+2                                             MTH00068
         SLJ      Q1Q02550                                          MTH00069
```

```
         ENTRY    Q1Q03500                                       MTH00070
Q1Q03500 SLJ      **               SUBTRACT AN INTEGER CONSTANT  MTH00071
         LDA      *                  FROM EACH WORD OF THE       MTH00072
         RTJ      INITIAL            ACCUMULATOR AND LOAD THE    MTH00073
         NOP                         A-REGISTER WITH A NON-ZERO  MTH00074
+        SAL      D1                 PART OF THE DIFFERENCE      MTH00075
         SAU      D2                 IF IT EXISTS OTHERWISE      MTH00076
         SAL      D3                 LOAD THE A-REGISTER WITH ZEROMTH00077
         SIU    1 D4                  (-OPERATOR)                MTH00078
         ENI    1 0                                              MTH00079
D1       LDA      ACC                                            MTH00080
         SUB      **                                             MTH00081
+        AJP      *+1                                            MTH00082
         ENI    1 0                                              MTH00083
+        STA      ACC                                            MTH00084
         LDA      ACC+1                                          MTH00085
D2       SUB      **                                             MTH00086
         STA      ACC+1                                          MTH00087
+        AJP      *+1                                            MTH00088
         ENI    1 1                                              MTH00089
D3       LDA      ACC+2                                          MTH00090
         SUB      **                                             MTH00091
+        AJP      *+1                                            MTH00092
         ENI    1 2                                              MTH00093
+        STA      ACC+2                                          MTH00094
         LDA    1 ACC                                            MTH00095
D4       ENI    1 **                                             MTH00096
         SLJ      Q1Q03500                                       MTH00097
```

```
         ENTRY     Q1Q03550                                             MTH00098
Q1Q03550 SLJ       **                 CLEAR BITS OF THE ACCUMULATOR     MTH00099
         LDA       *                     WHERE THERE ARE CORRESPON-     MTH00100
         RTJ       INITIAL               DING 1-BITS IN A LOGIC5        MTH00101
         NOP                             WORD .LOAD THE A-REGISTER      MTH00102
+        SAL       E1                    WITH A NON-ZERO PART OF        MTH00103
         INA       1                     THE RESULT IF IT EXISTS.       MTH00104
         SAU       E2                    OTHERWISE LOAD THE             MTH00105
         INA       1                     A-REGISTER WITH ZERO.          MTH00106
         SAL       E3                    (-OPERATOR)                    MTH00107
         SIU     1 E4                                                   MTH00108
         ENI     1 0                                                    MTH00109
E1       LDA       ACC                                                  MTH00110
         SCL       **                                                   MTH00111
+        AJP       *+1                                                  MTH00112
         ENI     1 0                                                    MTH00113
+        STA       ACC                                                  MTH00114
         LDA       ACC+1                                                MTH00115
E2       SCL       **                                                   MTH00116
         STA       ACC+1                                                MTH00117
+        AJP       *+1                                                  MTH00118
         ENI     1 1                                                    MTH00119
E3       LDA       ACC+2                                                MTH00120
         SCL       **                                                   MTH00121
+        AJP       *+1                                                  MTH00122
         ENI     1 2                                                    MTH00123
+        STA       ACC+2                                                MTH00124
         LDA     1 ACC                                                  MTH00125
E4       ENI     1 **                                                   MTH00126
         SLJ       Q1Q03550                                             MTH00127
```

```
         ENTRY    Q1Q04550                                            MTH00128
Q1Q04550 SLJ      **                  FORM THE LOGICAL PRODUCT        MTH00129
         LDA      *                      OF THE ACCUMULATOR AND       MTH00130
         RTJ      INITIAL                A LOGIC5 WORD.               MTH00131
         NOP                             (*OPERATOR)                  MTH00132
+        SAU      F1                                                  MTH00133
         INA      1                                                   MTH00134
         SAL      F2                                                  MTH00135
         INA      1                                                   MTH00136
         SAU      F3                                                  MTH00137
         LDQ      ACC                                                 MTH00138
F1       LDL      **                                                  MTH00139
         STA      ACC                                                 MTH00140
F2       LDQ      ACC+1                                               MTH00141
         LDL      **                                                  MTH00142
         STA      ACC+1                                               MTH00143
         LDQ      ACC+2                                               MTH00144
F3       LDL      **                                                  MTH00145
         STA      ACC+2                                               MTH00146
         SLJ      Q1Q04550                                            MTH00147
```

```
         ENTRY      Q1Q05500                                             MTH00148
Q1Q05500 SLJ        **                 GENERATE A 1-BIT IN               MTH00149
         SIU     1  G3                    THE BIT POSITION OF            MTH00150
         ENA        0                     THE ACCUMULATOR CORRES-        MTH00151
         STA        ACC                   PONDING TO THE INTEGER         MTH00152
         STA        ACC+1                 ARGUMENT COUNTING FROM         MTH00153
         STA        ACC+2                 THE MOST SIGNIFICANT           MTH00154
         LDA        Q1Q05500              BIT POSITION                   MTH00155
         NOP                               (/OPERATOR)                   MTH00156
+        RTJ        INITIAL                                              MTH00157
         NOP                                                             MTH00158
+        SAU        *+1                                                  MTH00159
+        LDA        **                                                   MTH00160
         ENI     1  -1                                                   MTH00161
G1       INI     1  1                                                    MTH00162
         INA        -48                                                  MTH00163
         AJP     2  G1                                                   MTH00164
         INA        48                                                   MTH00165
         STA        =SG2                                                 MTH00166
         ENA        48                                                   MTH00167
         SUB        G2                                                   MTH00168
         SAL        *+1                                                  MTH00169
+        ENA        1                                                    MTH00170
         ALS        **                                                   MTH00171
         STA     1  ACC                                                  MTH00172
G3       ENI     1  **                                                   MTH00173
         SLJ        Q1Q05500                                             MTH00174
```

```
         ENTRY    Q1Q10050                                           MTH00175
Q1Q10050 SLJ      **                  STORE ZERO IN EACH             MTH00176
         LDA      *                     PART OF A LOGIC5 WORD.       MTH00177
         RTJ      INITIAL                                            MTH00178
         NOP                                                         MTH00179
+        SAU      H1                                                 MTH00180
         INA      1                                                  MTH00181
         SAL      H1                                                 MTH00182
         INA      1                                                  MTH00183
         SAU      H2                                                 MTH00184
         ENA      0                                                  MTH00185
H1       STA      **                                                 MTH00186
         STA      **                                                 MTH00187
H2       STA      **                                                 MTH00188
         SLJ      Q1Q10050                                           MTH00189
```

```
          ENTRY      Q1Q10550                                              MTH00190
Q1Q10550  SLJ        **                  STORE THE ACCUMULATOR IN          MTH00191
          LDA        *                     A LOGIC5 WORD.                  MTH00192
          RTJ        INITIAL                                               MTH00193
          NOP                                                              MTH00194
+         SAL        J1                                                    MTH00195
          INA        1                                                     MTH00196
          SAL        J2                                                    MTH00197
          INA        1                                                     MTH00198
          SAL        J3                                                    MTH00199
J1        LDA        ACC                                                   MTH00200
          STA        **                                                    MTH00201
J2        LDA        ACC+1                                                 MTH00202
          STA        **                                                    MTH00203
J3        LDA        ACC+2                                                 MTH00204
          STA        **                                                    MTH00205
          SLJ        Q1Q10550                                              MTH00206



INITIAL   SLJ        **                                                    MTH00207
          ALS        24                                                    MTH00208
          INA        -1                                                    MTH00209
          SAU        *+1                                                   MTH00210
+         ENA      7 **                                                    MTH00211
          SLJ        INITIAL                                               MTH00212
          END                                                              MTH00213
```