1974

Pragmatic aspects of action/rule matrix partitioning with respect to time-space considerations /

Rolf G. Pinckert
Lehigh University

PRAGMATIC ASPECTS OF ACTION/RULE MATRIX PARTITIONING WITH RESPECT TO TIME-SPACE CONSIDERATIONS

by

ROLF G. PINCKERT

A Thesis

Presented to the Graduate Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Industrial Engineering

Lehigh University

1974

CERTIFICATE OF APPROVAL

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science.

19 April 1974
Date

John W. Adams
Professor in Charge

Chairman of the Department
of Industrial Engineering

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to Dr. John Adams, of the Industrial Engineeing Department of Lehigh University, and especially to Mr. Jonas Rabin, of the Western Electric Engineering Research Center Staff, for their guidance and helpful suggestions. Mrs. Patsy Clemetson also deserves special thanks for her patience and effort in the typing of this manuscript.

Finally, I would like to express a special note of gratitude to my wife Sandra, for her continued encouragement during the course of this effort.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

ABSTRACT

Until now, no systematic approach has been developed with regard to time-space optimization of the action/rule matrix of a limited-entry decision table. Existing algorithms partition the action/rule matrix either to minimize storage requirements or response time. A new algorithm developed in this thesis provides a general approach in that both time and space considerations are used in the partitioning process.

The new algorithm embraces a more realistic approach to the partitioning process than that provided by existing algorithms. It is unrealistic to presume that all actions in the matrix will require an identical amount of storage. It is likewise unrealistic to assume that all decision rules are equally likely to be selected. For these reasons, a means is provided in the new algorithm to specify variable action weights and different rule probability distributions, for use in the partitioning process.

Finally, the new algorithm can generate a set of partitioned matrices for a given input matrix. There are two main reasons for generating such a set of matrices:

1. An improved solution over that produced by the single output usually results.

2. Since the resultant set of matrices exhibit varying time-space requirements, these results can be useful in determining possible time-space tradeoffs.

CHAPTER 1

FUNDAMENTALS OF DECISION TABLES AND PARTITIONING

Elements of Decision Tables

A basic structural unit of a decision table is the decision rule. It is a statement that stipulates the set of conditions which must be satisfied in order that a sequence of actions be performed. Hence, decision rules imply an if...then... relationship between unique sets of conditions, and the actions performed in response to those condition sets. Consider the following example: "If your net income is not more than $600 and you are the head of the household, use Tax Table II." This statement represents a decision rule, where the conditions are net income and financial responsibility, and the use of Tax Table II is the action to be performed in response to the conditions. Whenever we arrange the decision rules pertaining to a particular process as a set of combinations of conditions, we have created a decision table. A representative table is illustrated in Figure 1.1.

Each column in the matrix represents a decision rule, or simply, a rule. The condition and action stubs of the table describe the conditions to be considered (or tested) and the actions to be performed, respectively. We

can divide the table into two functional matrices; namely, the condition/rule matrix and the action/rule matrix. It is the partitioning of the action/rule matrix which is the major topic of this thesis.

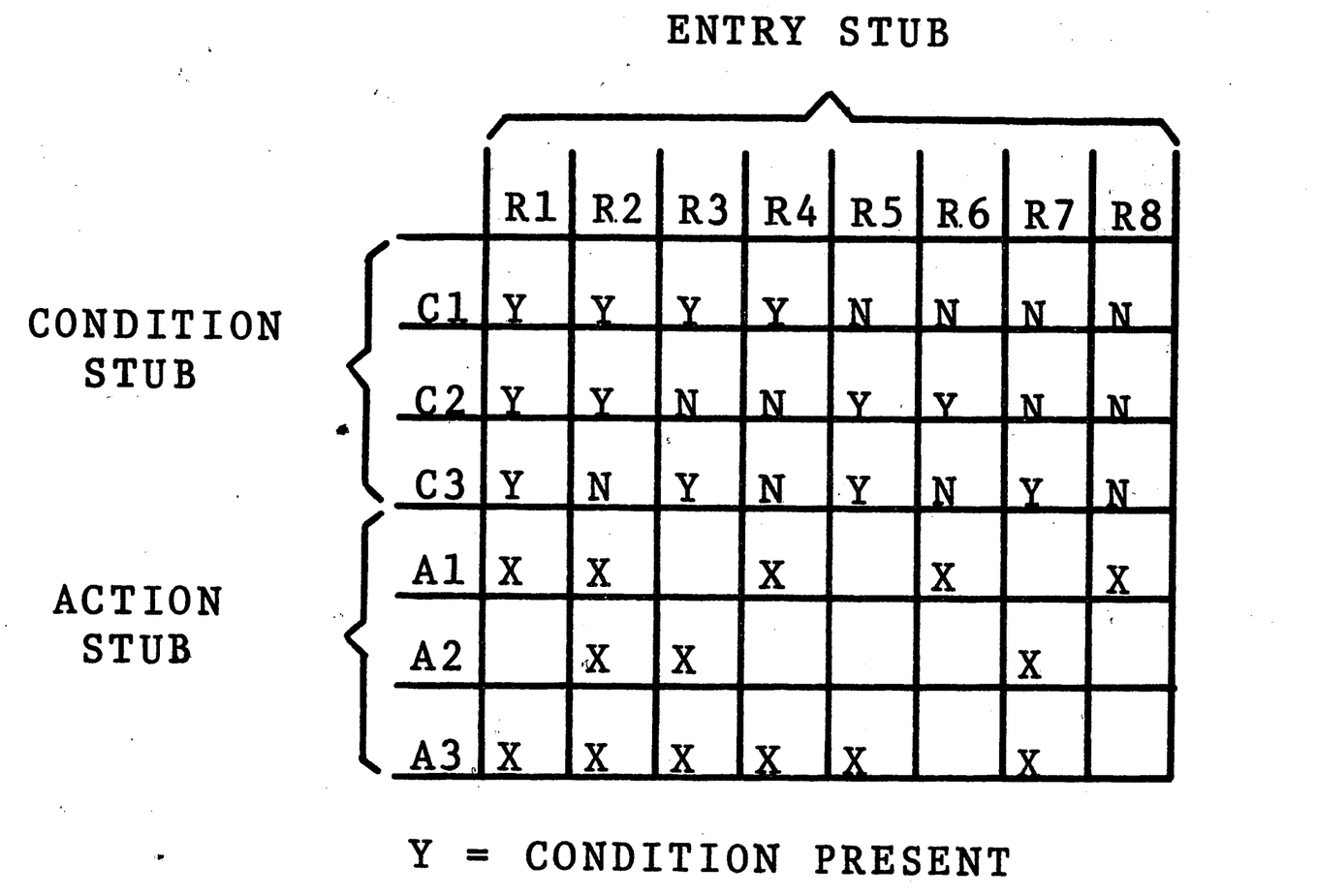


FIGURE 1.1

Action/Rule Matrix Partitioning Fundamentals

Since our emphasis will only be on the action/rule matrix, we will simply refer to it as "the matrix". It should be noted that the primary use of decision tables is in computer applications. As such, the tables will

place demands upon the resources of the user system, since they are converted into programmed statements which require a certain amount of storage and processor time for their execution. In the case of the action/rule matrix, this demand is a function of the number of statements associated with each action, and the time required to execute a path (or sequence of actions) through the matrix. Several techniques, which we will broadly classify as partitioning methods, have been developed to minimize these effects on the user system. There are basically three ways that the matrix can be partitioned: 1) By rows, 2) by columns, and 3) into smaller submatrices. The latter method is referred to as "blocking", and is the basis for the procedures to be described in this thesis.

To provide insight into blocking concepts, we begin with the following definition: A block is an identical sequence of two or more actions stipulated by at least two rules. Since the action sequences which form the block are identical, we can eliminate all but one without loss of information. Those sequences which can be eliminated obviously represent a reduction in the system memory required to store the matrix. We will refer to the required action sequence corresponding to a particular block as the block sequence, and the rules to which this

sequence is common as block rules. Furthermore, the action sequences which are specified by the block rules, and which remain to be performed upon exiting the block sequence, are called residual action sequences. Figure 1.2 illustrates these ideas.

	R1	R2	R3	R4
A1		X	X	
A2	X	X	X	
A3			X	X
A4				X
A5	X			X

BLOCK = {R2,R3;A1,A2}

BLOCK SEQUENCE = (A1,A2)

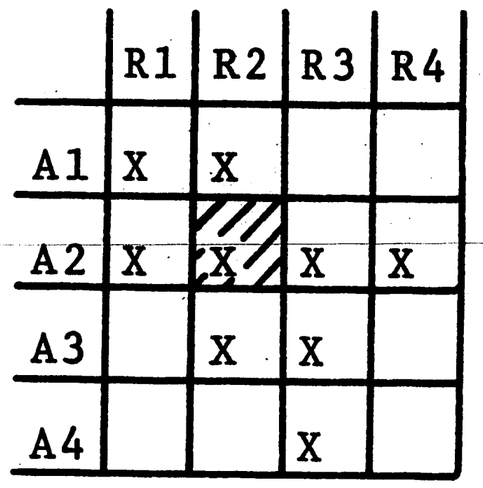
BLOCK RULES = {R2,R3}

RESIDUAL ACTION
SEQUENCE = EXIT (Rule 2)
A3, A4, EXIT (Rule 3)

FIGURE 1.2

There are several ways of classifying blocks. The first is the relationship of a given block to other blocks in the matrix. A block is independent if it does not overlap with any other blocks in the matrix, where an overlap is the common occurrence of at least one action

and one rule in both blocks. Figure 1.3 illustrates two overlapping blocks, where the overlap is indicated by the crosshatched area.



	R1	R2	R3	R4
A1	X	X		
A2	X	X	X	X
A3		X	X	
A4			X	

BLOCK 1 = {R1,R2;A1,A2}

BLOCK 2 = {R2,R3;A2,A3}

FIGURE 1.3

It is important to note here, that any resultant block set used to partition the matrix must contain only independent blocks. Therefore, whenever a potential block from an overlapping pair is selected, the overlapping portion cannot be consolidated into a block sequence.

Another means of classifying blocks is with respect to the type of branching logic required at the termination of the block sequence. A direct-transfer block is one which has no associated residual action sequences, a condition which arises if the given block immediately

preceeds another block or, if the given block sequence terminates at the exit point of the matrix. A conditional-transfer block has associated residual block sequences which differ by at least one entry. An example of each block type, and the associated flow diagram are illustrated in Figure 1.41 and 1.42, respectively. Note the action A_D in the block sequence of block 2 in Figure 1.42. This additional action is necessary to provide for branching to the proper residual action sequence upon termination of the block sequence.

	R1	R2	R3	R4	R5	R6
A1	X	X				
A2	X	X				
A3	X	X	X	X		
A4	X	X	X	X		
A5	X	X	X	X		
A6		X			X	X
A7	X		X		X	X
A8		X		X		

BLOCK 1 = {R5,R6;A6,A7}

BLOCK 2 = {R1,R2,R3,R4;A3,A4,A5,A_D}

BLOCK 3 = {R1,R2;A1,A2}

FIGURE 1.41

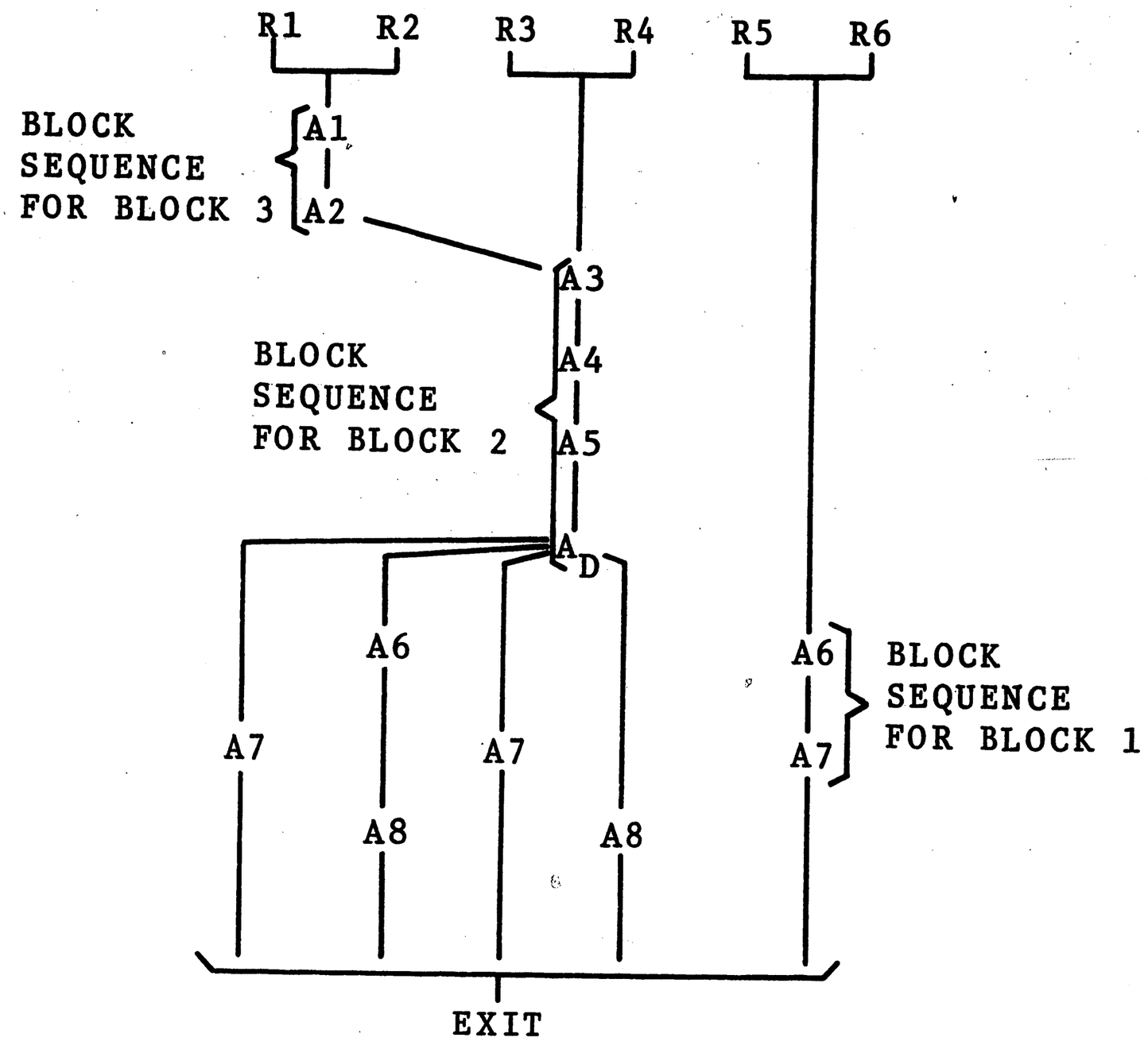


FIGURE 1.42

Form of the Action/Rule Matrix

Figure 1.5 illustrates the form of the action/rule matrix used in the development of our generalized blocking technique.

In the matrix of Figure 1.5, the execution of an action A_i for a particular rule R_j is specified by the presence of a one at location (i,j). Conversely, a zero at location (i,j) indicates that action A_i is not

to be executed for rule R_j. Furthermore, we assign the following characteristics to the matrix:

	P_1	P_2	P_3	...	P_{r-1}	P_r	
	R1	R2	R3		R_{r-1}	R_r	
A1	1	0	1	...	1	1	W_1
A2	0	0	1		0	1	W_2
	⋮				⋮		
A_{n-1}	0	1	0	...	1	1	W_{n-1}
A_n	1	1	0		0	0	W_n

FIGURE 1.5

(1) The action sequences for a given rule will be performed in the order specified in the matrix.

(2) An action can represent either a single statement or an entire program module.

(3) A weighting factor W_i is associated with each row i of the matrix, and is proportional to the memory storage requirements of the action A_i.

(4) Associated with each rule R_j of the matrix is the probability p_j, that the j^{th} rule is selected upon entry into the matrix, where

$$p_j \geq 0, \text{ and}$$

$$\sum p_j = 1, \quad j = 1, 2, \ldots, r$$

Current Blocking Algorithms

There are two blocking algorithms [2,7] presently in use which vary somewhat in the approach to partitioning the matrix. Because both techniques utilize the blocking concept, the resulting effect in either case is to maximize the efficient use of core memory. However, neither technique provides a means of measuring the contribution of each block towards the most efficient use of both time and space resource of the user system. In addition, both algorithms constrain all actions to be equally weighted, and all rules to be equally likely to be selected.

The initial application of blocking the action/rule matrix is the algorithm developed by Penick [7]. This procedure maximizes the number of direct-transfer blocks in the block set used to partition the matrix. This results in a reduction of redundant coding while minimizing the additional transfer logic required. However, priority given to direct-transfer blocks often precludes

the selection of larger conditional-transfer blocks in the matrix, which is a disadvantage when core memory is a premium resource.

Another approach to partitioning the matrix was developed by Burnham [2]. The basic philosophy of this method is to select the largest remaining block at each search through the matrix. Thus, the final result consists of a set of independent blocks which have been determined in decreasing order of dimension. We feel that the block selection method may lead to results which do not reflect the maximum savings obtainable. We will present a more detailed discussion of this subject in the next chapter.

Primary Features of the New Partitioning Module (NPM)

Like the Penick and Burnham algorithms just described, NPM also uses the blocking concept. Unlike these two algorithms, NPM is completely flexible with respect to the block selection criterion. It is our contention that the matrix should be partitioned according to the constraints of the environment in which it is to function. This implies that not only the resources of the user system, but also the characteristics of the process, which are inherent in the decision rules, must be considered in our approach to partitioning the matrix. We feel that it is totally unrealistic to assume that,

in general, the decision rules are equally likely to be selected. In addition, it is equally unrealistic to assume that the storage requirements of all actions are identical. To overcome these limitations of existing algorithms, we have incorporated the following features into NPM:

(1) A means of monitoring and updating rule frequencies based on actual usage, and

(2) Action weights that are proportional to the number of statements contained in the associated action.

A major emphasis in the development of NPM is to provide a means of determining both time and space requirements of a partitioned matrix. This is in contrast to existing algorithms which address either the time or the space aspect. If all of the blocks which are to form partitions of the input matrix are selected on the basis of a specified, invariant, selection criterion (as they are in existing algorithms), then for a given input matrix, only one specific partitioned matrix can result. However, if we examine the binary input matrix of Figure 1.6, we can identify several alternative ways of partitioning the input matrix, as shown.

	R1	R2	R3	R4
A1	1	1	0	0
A2	1	1	1	1
A3	1	1	1	1
A4	0	1	1	1

INPUT MATRIX

	R1	R2	R3	R4
A1	1	1	0	0
A2	1	1	3	3
A3	1	1	3	3
A4	0	2	3	3

ALTERNATIVE 1

	R1	R2	R3	R4
A1	1	3	0	0
A2	2	2	2	2
A3	2	2	2	2
A4	0	4	5	6

ALTERNATIVE 2

	R1	R2	R3	R4
A1	1	1	0	0
A2	1	1	4	5
A3	2	3	3	3
A4	0	3	3	3

ALTERNATIVE 3

FIGURE 1.6

If we were to examine all possible partitioned matrices that can be generated from a given binary matrix, and determine the time-space requirements of each, we could identify that result which optimally satisfies the specified criterion. This total enumeration approach requires a substantial amount of computing time to search all selection levels (first block selection, second block selection, etc.) of a large, complex matrix. However, results of our investigations indicate that complete enumeration is unnecessary to obtain results which are improvements over those of existing algorithms.

Since we have determined that alternative partitioned matrices can be generated from a given input matrix (i.e., Figure 1.6), it would be useful to provide machine capability for generating such a set of matrices and determine how each compares to the others of the set in terms of time-space demands on the user system. This implies that some degree of independence from the prescribed selection criterion is required. In the case of NPM, this independence from the selection criterion is provided at the zero level (i.e., first block selection), in that a source of alternative blocks is provided. This set consists of all blocks which can be identified in the original input matrix, and is called the zero-level block set B^0. Thus, by specifying a zero-level block initially and basing

subsequent selections on the specified criterion each time we partition the matrix, we can generate as many partitioned matrices as there are entries in $B_{\cdot}^{0}$. For example, if

$$B^{0} = \{B_1^0, B_2^0, \ldots, B_N^0\} ,$$

then N partitioned matrices can be generated. This concept is illustrated in Figure 1.7.

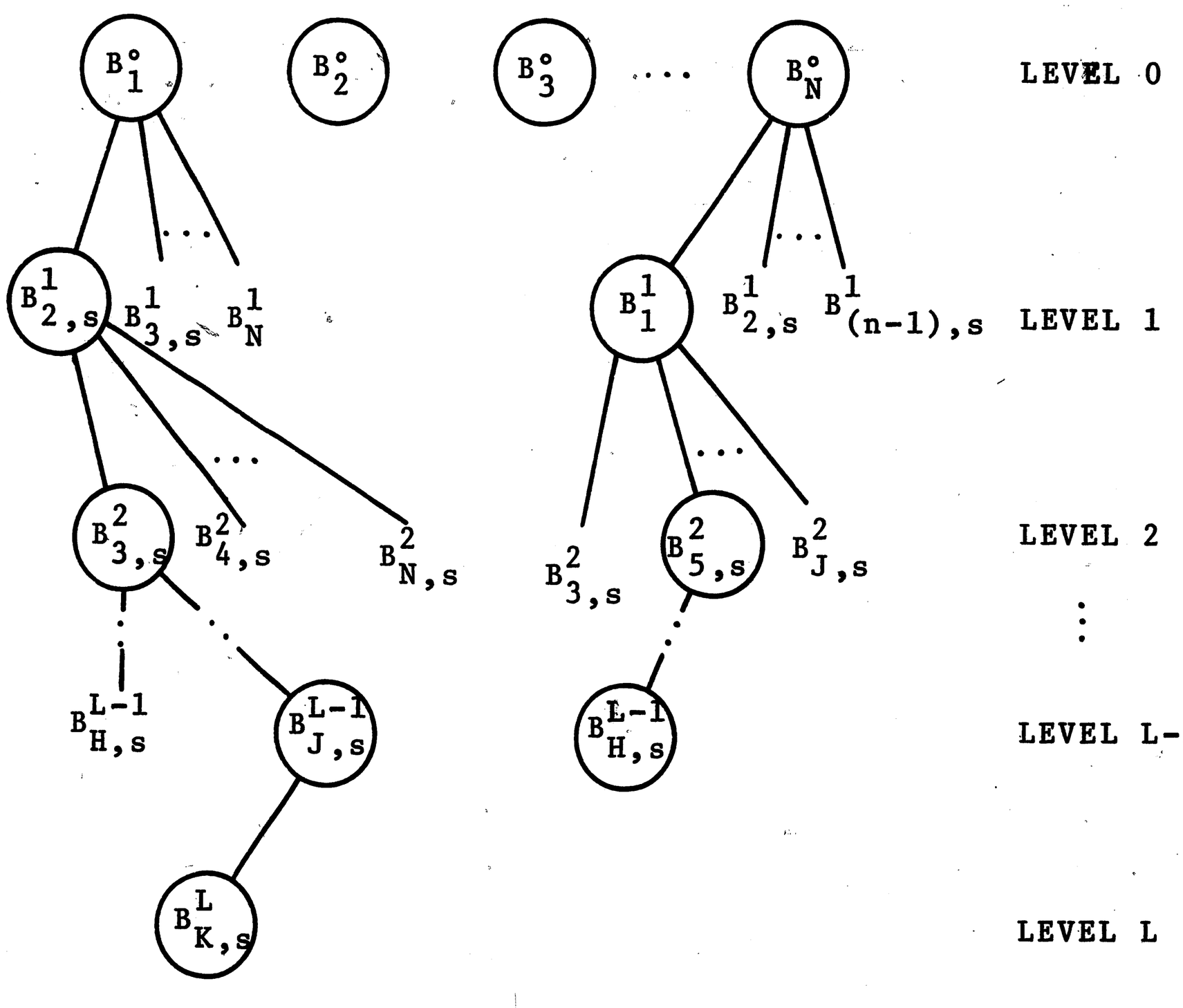


FIGURE 1.7

In Figure 1.7, the superscript refers to the selection level, the first subscript indicates the zero-level block designation, and the subscript s indicates that the block is a subset of the same block on a previous level. The block selections at each level are identified by circles. For example, if block 1 is specified as the first block selection, the resultant block set $(B_1^0, B_{2,s}^1, B_{3,s}^2, \ldots, B_{J,s}^{L-1}, B_{K,s}^L)$ will be used to partition the matrix. Obviously, a set of matrices generated by specifying different zero-level blocks is only a subset of all partitioned matrices that can be generated from a given input matrix. However, since zero-level blocks, are, in general, much larger (in total entries) than blocks at subsequent levels, these blocks will have the most effect on the time-space requirements of a partitioned matrix.

We have incorporated into NPM, the capability (a user option) of generating a set of alternative partitioned matrices for two reasons. First, in spite of an established block selection criterion, we have no guarantee that the resultant partitioned matrix is optimal or near optimal. By generating this set of alternative matrices, we increase the likelihood that an optimal or near optimal solution has been generated.

Second, it is felt that such a set of matrices which exhibit varying time-space requirements, will provide the designer with a useful tool for decision table software development, in that time-space tradeoffs can be achieved, which best satisfy design constraints.

CHAPTER 2

DETERMINATION OF THE EFFECTS OF BLOCKING ON TIME AND SPACE

Measurement of Block Contribution to Efficient Utilization of Space

We stated previously, that the consolidation of action sequences is a memory saving technique. For minimization of space requirements, the question of "how much" memory can be saved by incorporating a particular block is of obvious importance. Therefore, we must formulate a measure of the benefit derived from the formation of a particular block.

Consider a block composed of k actions and m rules. Because the action sequences forming the block are identical for the m rules, we can eliminate all but one of these action sequences without loss of information. This implies that we can save the space required to store the k(m-1) unnecessary actions if we incorporate the block. Thus, the quantity k(m-1) represents the inherent space savings of the block. In contrast, Burnham's [2] selection criterion incorporates the block having the largest total number of actions. In the case of overlapping blocks which have the same total number of actions, the block having the largest number of actions

in its block sequence is selected. As we will show in the following example, this procedure does not guarantee that the selected block provides the maximum savings in storage.

Consider the two potential blocks indicated in the matrix of Figure 2.1.

	R1	R2	R3
A1	X	X	
A2	X	X	X
A3	X	X	X

Block 1 = {R1,R2;A1,A2,A3}

Block 2 = {R1,R2,R3;A2,A3}

Figure 2.1

If we assume that all actions have a unit weight then, using Burnham's criteria for block selection, we would incorporate block 1 since both blocks have the same total number of actions (6), and block 1 contains the largest number of actions in its block sequence (3 vs. 2 for block 2). We can determine the memory savings (in terms of actions) inherent in each block directly from Figure 2.2, where the symbol ± indicates that the

Corresponding action does not require programming.

BLOCK 1	BLOCK 2
X ±	X X
X ± X	X ± ±
X ± X	X ± ±

Figure 2.2

From this simple example, we can readily see that block 2 actually results in a larger reduction in memory requirements than is possible with block 1.

With these ideas in mind, we can formulate an expression for the potential storage reduction, PSR, obtainable from a block as follows: For a block consisting of m rules and k actions,

$$PSR = (m-1)\sum_{k} W_k, \qquad (I)$$

where k indexes the actions in the block sequence, and W_k is the weight assigned to the k-th action of the block sequence. Equation (I) provides a more realistic approach to measuring storage reduction than existing methods, as we no longer restrict the actions of the matrix to be equally weighted.

Effect of Conditional-Transfer Blocks on Potential Storage Reduction

We indicated in Chapter 1 that conditional-transfer blocks require additional logic to provide for proper

branching upon termination of the block sequence. Since this decision logic is associated with all of the block rules, it effectively appears as an additional action, A_D, in the block sequence. If we assign a weight W_D to this decision node, we can express the net storage reduction, NSR, as

$$NSR=-W_D+(m-1)\sum_k W_k \ . \qquad \text{(II)}$$

Effect of Conditional-Transfer Blocks on Expected Execution Time.

As indicated in the previous section, the decision logic of a conditional-transfer block is common to all block rules. Therefore, the increase in the expected execution time, EET, of the matrix due to the formation of the conditional-transfer block is,

$$EET = T_D\sum_m p_m \ , \qquad \text{(III)}$$

where T_D is the number of time units required to execute the decision logic, m indexes the block rules, and p_m is the frequency of execution of the m-th block rule. Equations (II) and (III) provide us with a means of measuring the effect incorporating a particular block will have on the resources of the user system.

Integration of Time and Space Considerations

Whenever we are faced with software design, we are

usually constrained by the amount of time allowed for the program to execute and/or the amount of storage that the program can use. Unfortunately, a reduction in storage requirements usually results in a corresponding increase in execution time, and vice versa. By restructuring the basic programming scheme, we can generate a series of programs which provide identical results, but place varying demands upon system resources. Beizer [1] refers to this set of alternative programs as a "family" of programs. If we were to determine the space-time requirements of each program in the family, we could produce a space-time plot similar to the example illustrated in Figure 2.3.

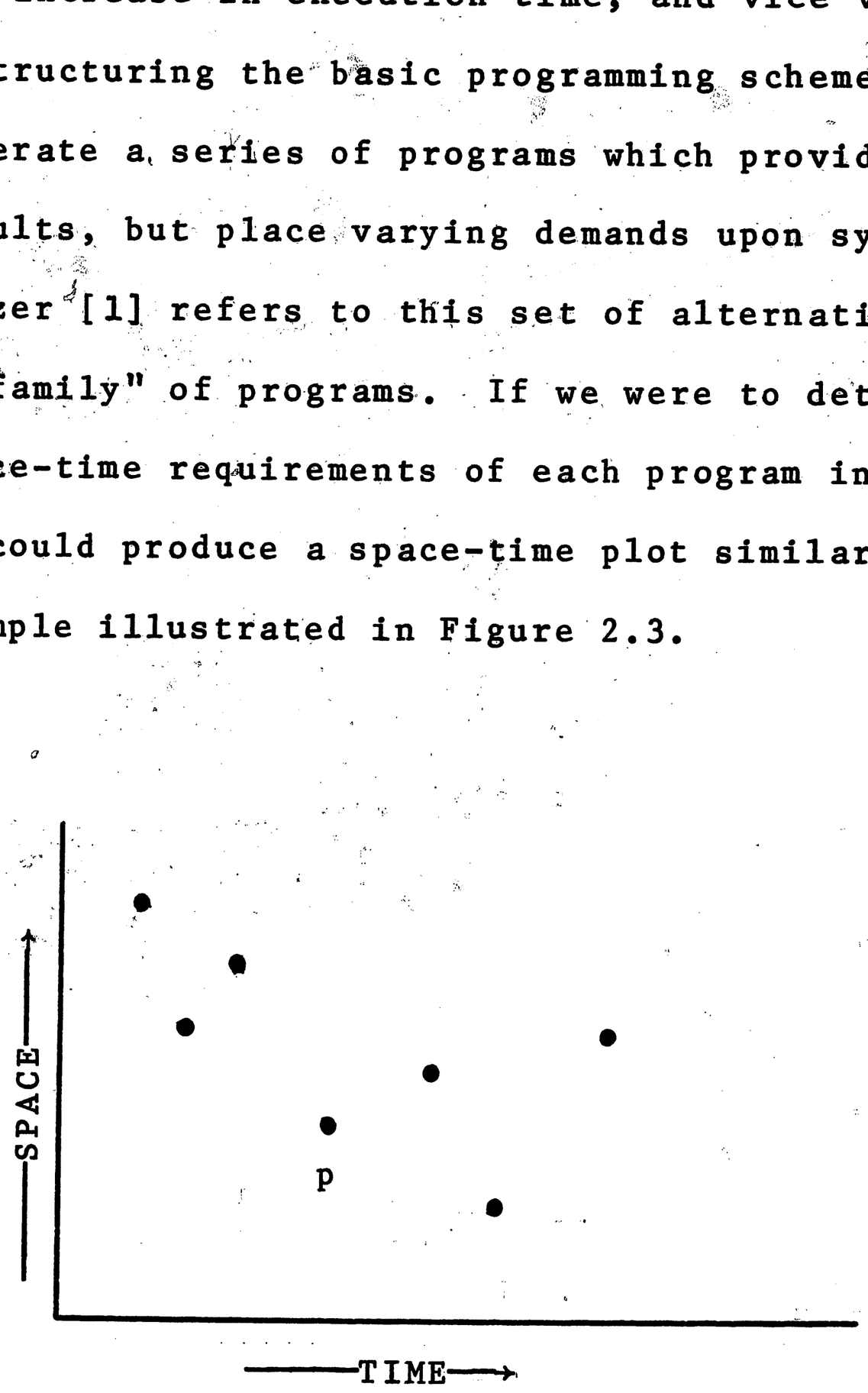


FIGURE 2.3

From this plot, we can determine the most efficient programs by identifying the undominated entries of the set. The region of dominance of a program P is defined as those points located in the first quardrant which has the space-time coordinate location p of the program P as its origin. Those entries in Figure 2.3 which are not in the region of dominance of any other entry, represent the most efficient programs.

If we now consider the fact that a partitioned matrix corresponds to a unique flowchart, then any alternative partitioning of the matrix will result in a different flowchart. Thus, like the family of programs described by Beizer, we can generate a family of flowcharts, each of which has associated time-space requirements. If we generate the time-space coordinates corresponding to the various partitioning alternatives, and determine the undominated entries, we will have identified the most efficient partitionings of the matrix.

The procedure we will describe in the following chapters will provide us with the capability of generating the time-space coordinates corresponding to alternative partitioned matrices. We feel that this result will provide the designer with the flexibility he needs to select that partitioning scheme which best fits his requirements.

CHAPTER 3

DESCRIPTION OF THE NEW PARTITIONING MODULE (NPM)

Introduction to the Technique

The function of NPM is to transform a binary input matrix into either a single partitioned matrix or a set of partitioned matrices, and to provide the analyst with the time-space requirements of each such partitioned matrix. As illustrated in Figure 3.1, the module inputs consist of the binary input matrix and the associated rule probabilities and action weights. The partitioning technique is divided into two stages: 1) the identification stage, during which all zero-level blocks existing in the input matrix are determined, and 2) the selection-modification stage, during which the sequential selection of blocks forming the partitions, and the modification of the remaining blocks after each block selection are carried out. Either a single partitioned matrix, where all block selections are based on a specified criterion (i.e., maximize the net storage reduction or minimize the increased expected execution time), or a set of partitioned matrices (multiple output), where each matrix of the set is partitioned in response to a different zero-level block selection, can be generated by NPM at the users discretion. We will provide greater detail with respect to the partitioning process in subsequent section of this chapter.

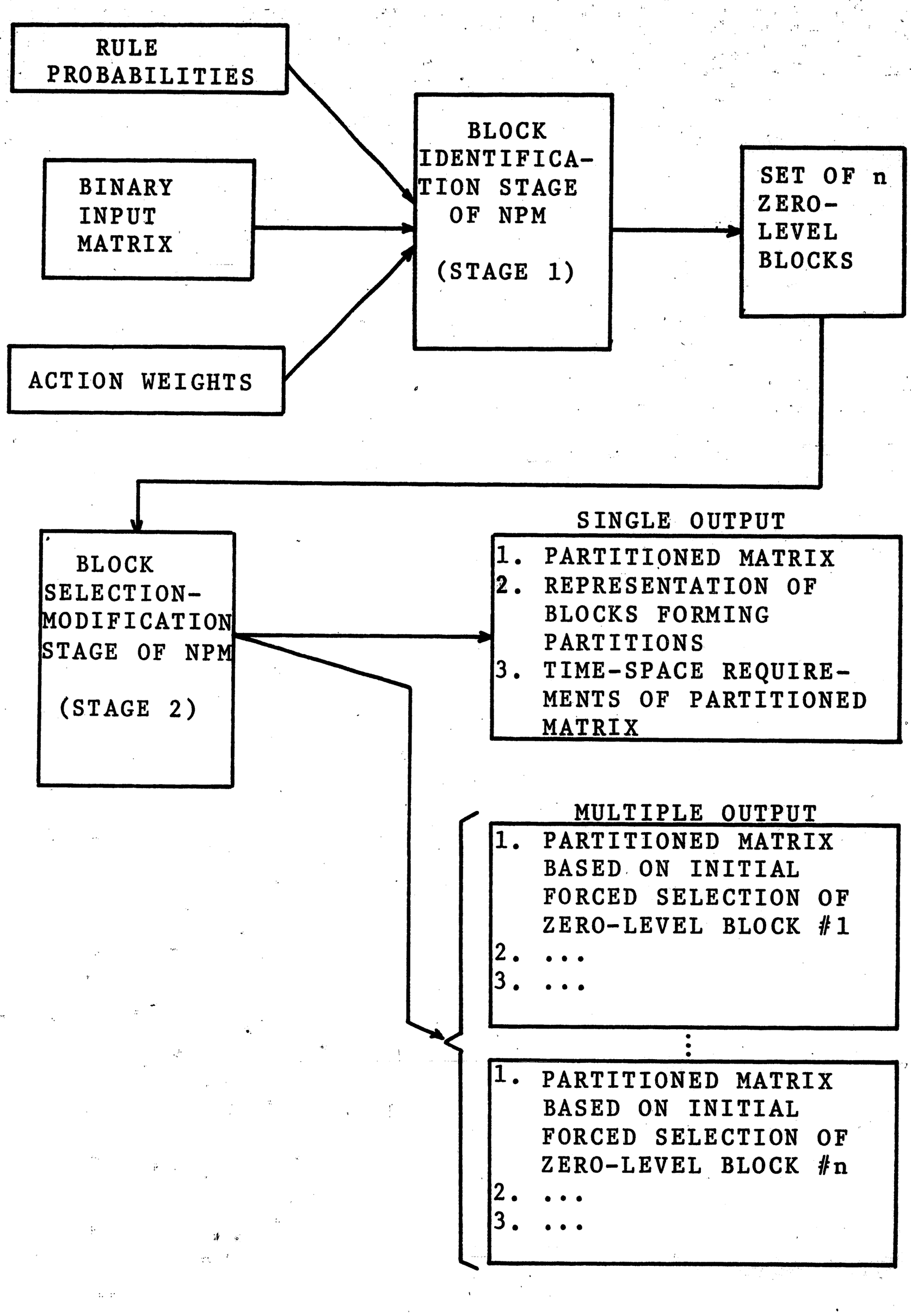

RULE PROBABILITIES
BINARY INPUT MATRIX
ACTION WEIGHTS
BLOCK IDENTIFICATION STAGE OF NPM
(STAGE 1)
SET OF n ZERO-LEVEL BLOCKS
BLOCK SELECTION-MODIFICATION STAGE OF NPM
(STAGE 2)
SINGLE OUTPUT
1. PARTITIONED MATRIX
2. REPRESENTATION OF BLOCKS FORMING PARTITIONS
3. TIME-SPACE REQUIREMENTS OF PARTITIONED MATRIX
MULTIPLE OUTPUT
1. PARTITIONED MATRIX BASED ON INITIAL FORCED SELECTION OF ZERO-LEVEL BLOCK #1
2. ...
3. ...
1. PARTITIONED MATRIX BASED ON INITIAL FORCED SELECTION OF ZERO-LEVEL BLOCK #n
2. ...
3. ...

FIGURE 3.1

Input Requirements

There are three inputs required by NPM:

1. The binary action/rule matrix,
2. an action weight vector whose entries are proportional to the storage requirements of the corresponding action, and
3. a probability vector whose entries are the probabilities of the rules being executed.

Since rule probabilities are used as a basis for block selection, it is highly desirable to obtain accurate estimates of their values. These estimates can be obtained by using a diagonal square matrix of dimension equal to the number of rules in the original decision table, as shown in Figure 3.2. Each element on the main diagonal corresponds to a unique rule, and each action $b_i, i=1,\ldots,r$, records the number of times the associated rule is executed during a prescribed number of entries into the table. Using this scheme, the rule probabilities can be reexamined periodically, say every N entries into the table, to determine if a significant change has occurred in their distribution. If a significant difference is detected, the new probabilities can be computed by,

$$p_j = \frac{b_j}{N}, \quad j=1,2,\ldots,r.$$

The matrix can then be repartitioned using the updated probabilities.

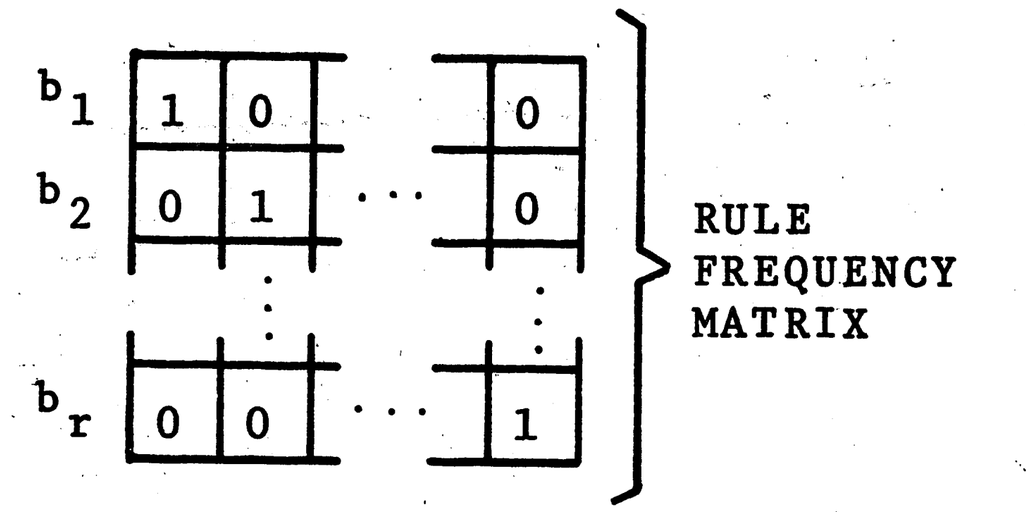


FIGURE 3.2

The Block Identification Stage

It is the function of the Block Identification Stage to determine the zero-level blocks in the input matrix. To identify these zero-level blocks, the binary matrix is converted into a cumulative-sum matrix. A description of this procedure follows, and an illustrative example is presented in Figure 3.3.

STEP 1 Assign to each row of the binary matrix an increasing power-of-two, assigning 2^0 to the last row, 2^1 to the next-to-last row, etc.

STEP 2 Starting with the next-to-last row, replace all 1 entries with the sum of the assigned row value and the corresponding entry in the last row.

STEP 3 Repeat this procedure for all preceeding rows, replacing all 1 entries with the sum of the row value and the previous cumulant in the column corresponding to the 1 entry. Note that the zero-entries of the matrix are not altered.

The cumulants formed in this manner are unique. Thus, if identical cumulants are encountered in a particular row of the cumulative-sum matrix, we know that the corresponding entries for the remainder of the columns associated with these cumulants are also identical. As the zero-level blocks are identified, they must be maintained in a form which will readily provide information regarding block interrelationships, and that can easily be modified in response to a particular block selection. This requirement

	R1	R2	R3
A1	1	0	1
A2	0	1	1
A3	1	1	1

a) BINARY INPUT MATRIX

	R1	R2	R3
A1	1	0	1
A2	0	1	1
A3	1	1	1

b) ASSIGN POWERS-OF-TWO

	R1	R2	R3
A1	1	0	1
→ A2	0	3	3
A3	1	1	1

c) BEGIN FORMING CUMULATIVE-SUM MATRIX

	R1	R2	R3
→ A1	5	0	7
A2	0	3	3
A3	1	1	1

d) COMPLETE CUMULATIVE-SUM MATRIX

FIGURE 3.3

is met by utilization of block vectors. A block vector is a storage array containing information about a particular block in the matrix. The syntactical representation of a block vector is given as follows:

<BLOCK VECTOR>::=<ACTION FIELD><RULE FIELD><MISCELLANEOUS SUBFIELD #1><MISC. SUBFIELD #2><MISC. SUBFIELD #3>

where,

<ACTION FIELD> contains 1-entries corresponding to the row indices (action #) of the block.

<RULE FIELD> contains 1-entries corresponding to the column indices (rule #) of the block.

<MISC. SUBFIELD #1> contains; 0=direct-transfer block
1=conditional-transfer block
during block identification, or
1=overlapping block
2=zero(empty) vector
during block selection-modification.

<MISC. SUBFIELD #2> contains the number of actions in the block.

<MISC. SUBFIELD #3> contains the number of rules in the block.

As an example of how a block is represented in block vector form, consider the binary input matrix of Figure 3.41, which has been located during the block-identification stage. The vector entries for this case are illustrated in Figure 3.42.

	R1	R2	R3
A1	1	1	1
A2	0	1	1
A3	0	0	1

BLOCK = {R2,R3;A1,A2}

FIGURE 3.41

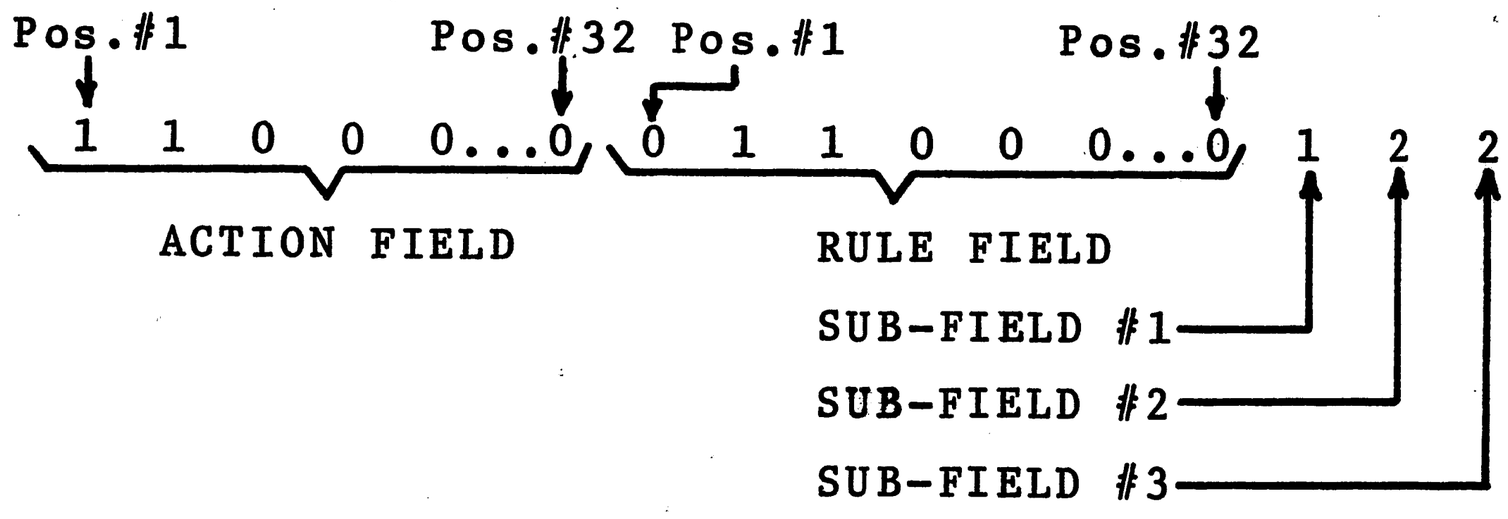


FIGURE 3.42

In figure 3.42, there are 1-entries in positions one and two of the action field, corresponding to the rows (actions) in the block, and 1-entries in positions two and three of the rule field, corresponding to the columns (rules) in the block. Subfield one contains a 1 since we are in block identification and the block is obviously a conditional-transfer block. The 2 in subfield 2 represents the number of actions in the block and the 2 in subfield 3 is the number of rules in the block.

The zero-level block vectors and the associated NSR (net storage reduction) and EET (expected execution time) values are stored in the IAR Table and VNET Table, respectively, during the block identification stage as is illustrated in Figure 3.5.

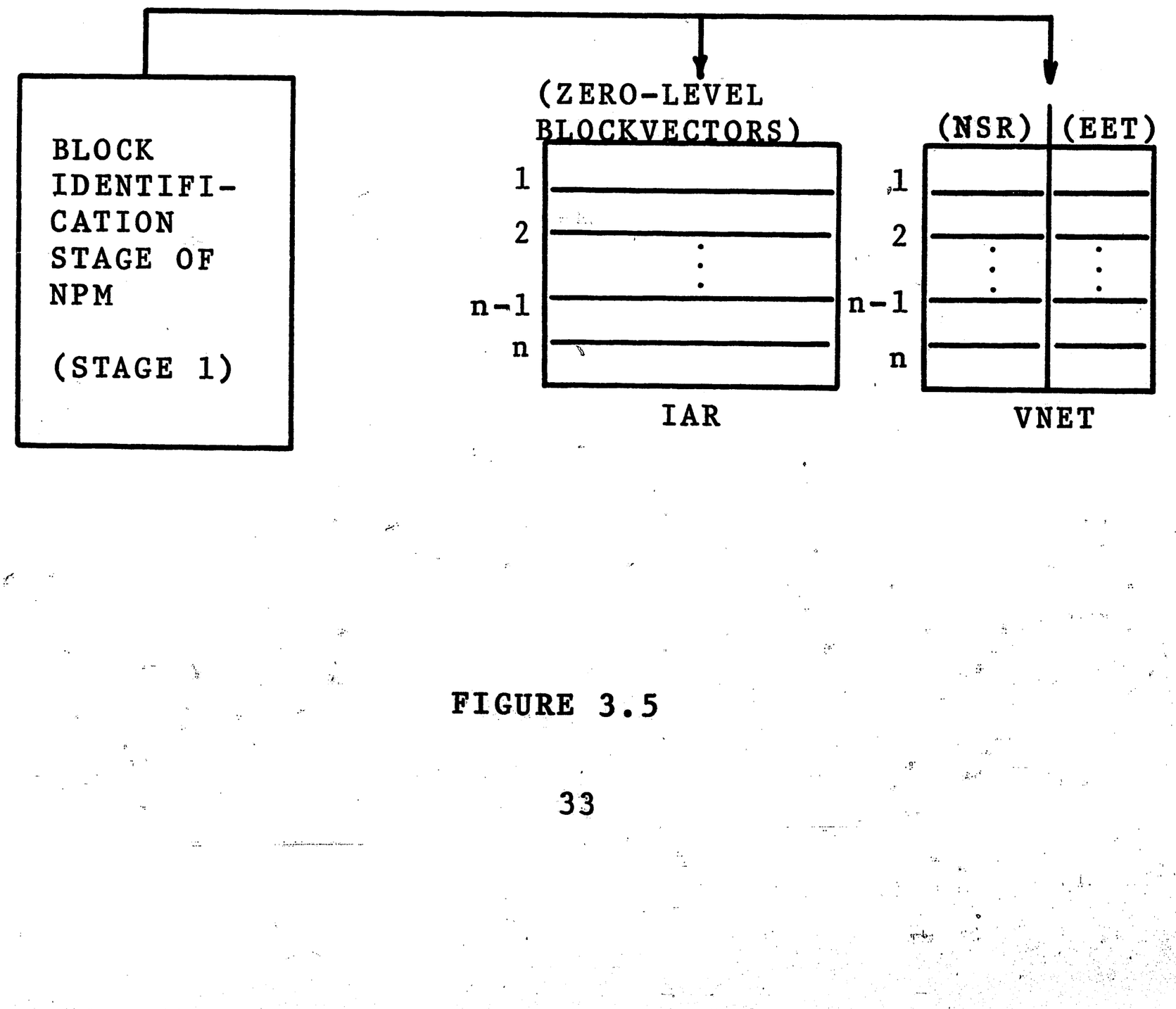


FIGURE 3.5

Flowchart 1 of Appendix B provides a detailed description of the procedure used in identifying the zero-level blocks in the cumulative-sum matrix described earlier in this section.

The Block Selection-Modification Stage

It is the function of the selection-modification stage to determine the partitions of the binary input matrix. To accomplish this, the block remaining in the matrix which best satisfies the specified block selection criterion (either maximization of NSR or minimization of EET) is incorporated as a partition of the input matrix. The remaining blocks of the input matrix which overlap the selected block are then modified in response to this selection. This insures that the modified blocks are independent of the selected block. As illustrated in Figure 3.6, the IARB Table and the VNETB Table are used to store the block vectors and the corresponding NSR and EET values, respectively, resulting during the selection-modification process. As illustrated in Figure 3.6, the IARB Main Sub-table is initialized from the IAR Table at the start of each partitioning run. At this point, the Main Sub-table will contain all zero-level blocks in the action/rule matrix.

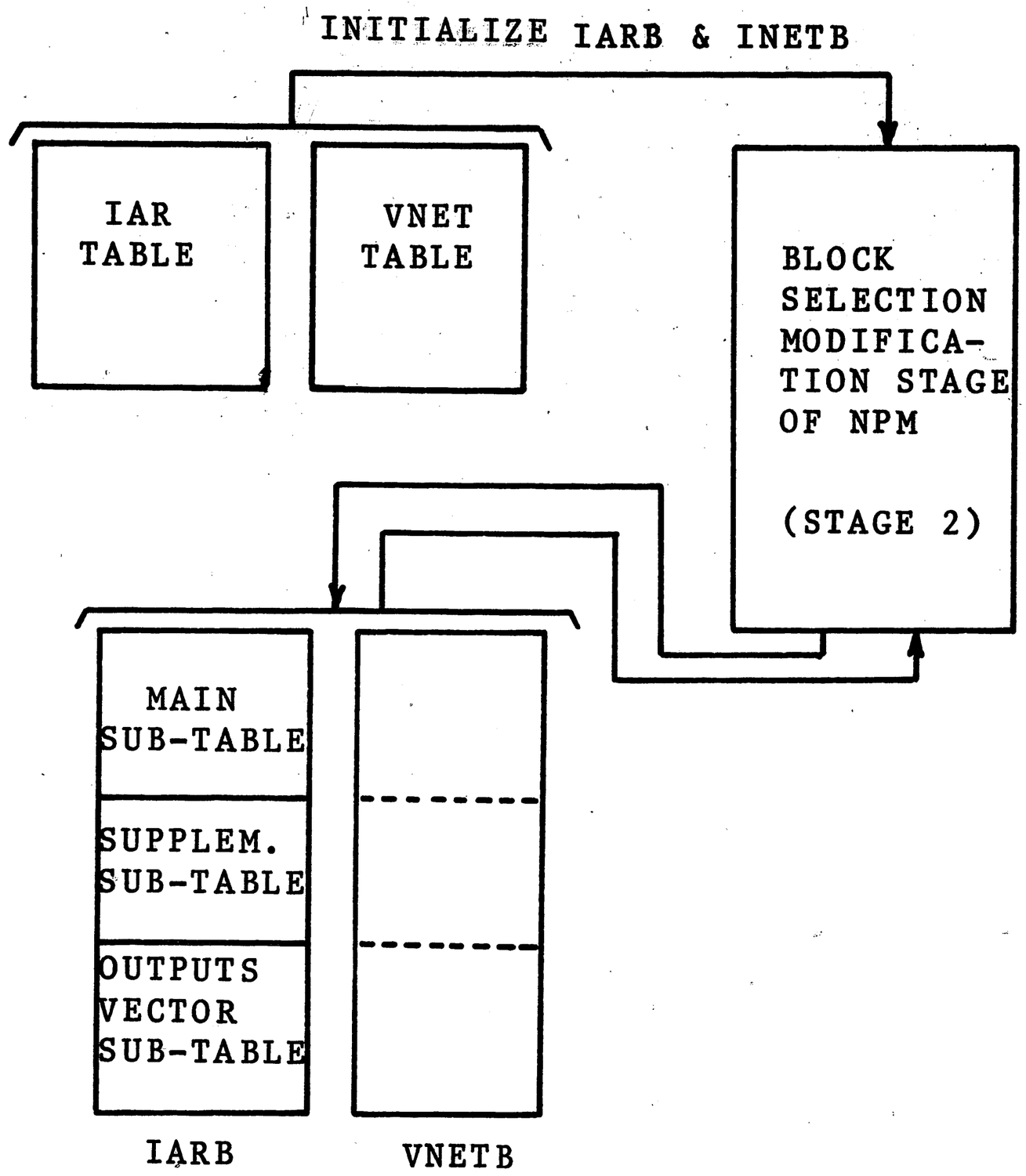


FIGURE 3.6

The initial block selection can be accomplished in one of two ways:

1. Based on one of the available selection criteria indicated above, or

2. a forced selection during a multiple output run.

Regardelss of the type of initial selection, all subsequent selections are based on the specified criterion. The following discussion will elaborate on the steps involved in a typical selection-modification cycle.

Consider the set of blocks illustrated in Figure 3.71 and the entries in the IAR Table as shown in Figure 3.72.

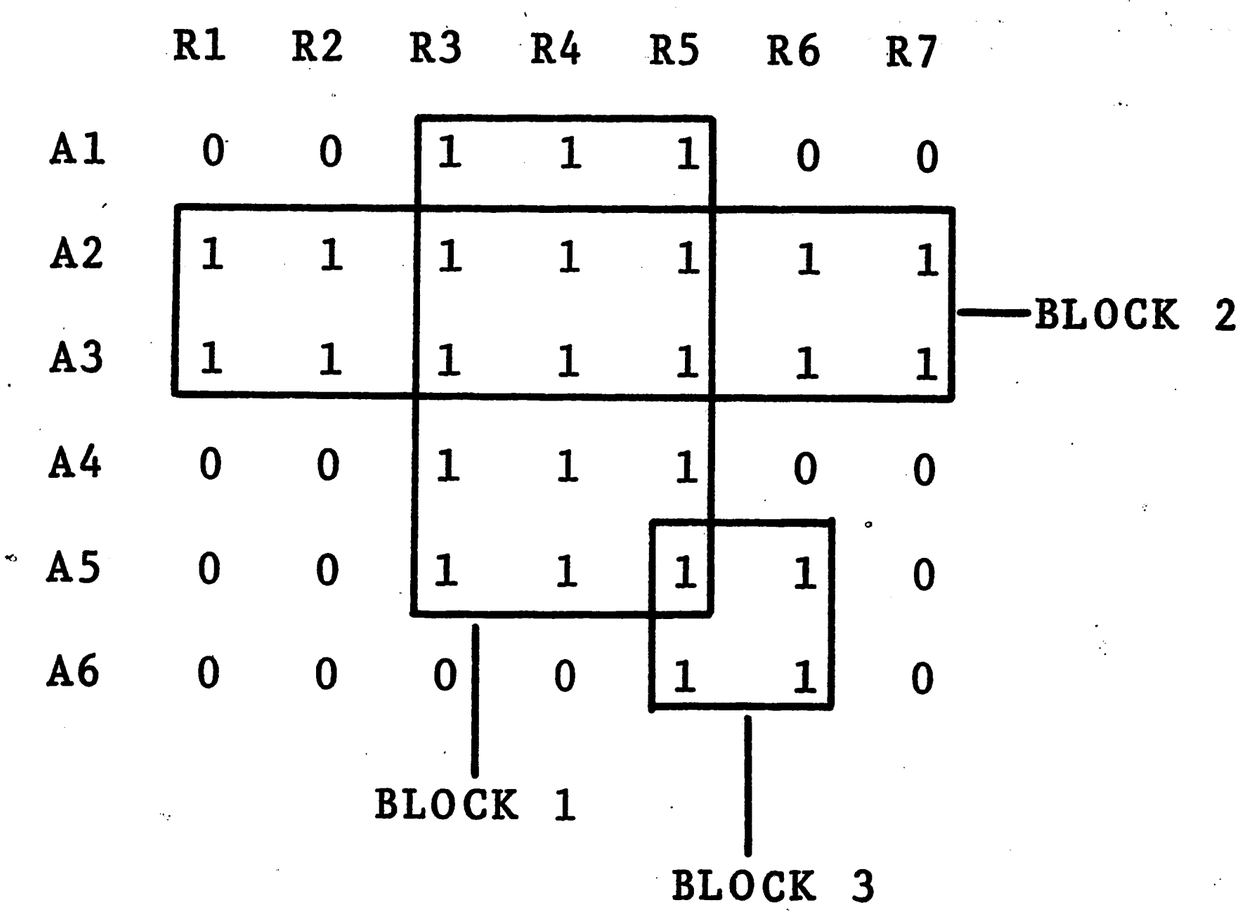


FIGURE 3.71

IARB (MAIN)(PRE-SELECTION STATE)

POSITION→1	32 1	32	VNETB NSR	EET
BLOCK 1	1 1 1 1 1 0...0	0 0 1 1 1 0 0...0	1 5 3 9.	0.429
BLOCK 2	0 1 1 0 0 0...0	1 1 1 1 1 1 1...0	1 2 7 11.	1.0
BLOCK 3	0 0 0 0 1 1...0	0 0 0 0 1 1 0...0	0 2 2 2.	0.0

FIGURE 3.72

For this example the specified selection criterion will be to maximize NSR. Each action will be assumed to have a weight of one, and all rules are assumed equally probable.

Since selection is based on the maximum NSR value, block 2 would be transferred to the Output Vector Sub-table. A snapshot of the post-selection Main and Output Vector Sub-tables is illustrated in Figure 3.73.

IARB (MAIN)-(POST-SELECTION STATE)

POSITION→	1						32	1						32			VNETB NSR	EET
BLOCK 1	1	1	1	1	1	0	0	0	1	1	1	0	0	0	1	5	3	9. 0.429
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0. 0.0
BLOCK 3	0	0	0	0	1	1	0	0	0	0	0	1	1	0	1	2	2	2. 0.0

IARB (O.V.)-(POST-SELECTION STATE)

POSITION→	1						32	1						32			VNETB NSR	EET
	0	1	1	0	0	0	0	1	1	1	1	1	1	0	1	2	7	11.0 1.0

FIGURE 3.73

At this point the blocks in the input matrix which overlap with the selected block must be modified by at least the amount of the overlap. There are two consequences of modifying an existing block:

1. The modified block is eliminated in that there are insufficient non-overlapping action sequences

remaining to form a legitimate block.

2. A new block(s) is(are) generated from the non-overlapping action sequences remaining after the modification.

Note that condition 1 applies to block 3 and condition 2 applies to block 1. The post-modification IARB (Main) Table is illustrated in Figure 3.74.

IARB (MAIN)-(POST-MODIFICATION STAGE)

POSITION	1						32	1						32	VNETB NSR	VNETB EET
BLOCK 1	0	0	0	1	1	0	...0	0	0	1	1	1	0	0...0 1 2 3	3.	0.429
	0	0	0	0	0	0	...0	0	0	0	0	0	0	0...0 2 0 0	0.	0.00
	0	0	0	0	0	0	...0	0	0	0	0	0	0	0...0 2 0 0	1.	0.00

FIGURE 3.74

In general, for two overlapping blocks M and N, if block M is selected, block N can no longer exist in its original form because the entries common to both blocks will be incorporated in block M. However, the actions of block N which are not common to block M, become "free" actions which can form new blocks. The generation of a new block can occur in two ways:

1. If identical action sequences consisting of at least two entries are freed by the elimination of the source block N, and these action sequences

occur in two or more rules exclusive of the block rules of M, then these sequences can combine to form a new block. We refer to this type of modification as rule modification.

2. If two or more identical action sequences containing at least two entries are freed by the elimination of the source block N, and the rules corresponding to these sequences intersect the set of block rules of M, but are exclusive of the block sequence of M, then these sequences can combine to form a new block. We refer to this type of modification as action modification.

Each vector in the Main Sub-table is subjected to both rule and action modification (if applicable) by NPM. Since this procedure can generate more than one new block from a given block vector, the Supplementary Sub-table is used to provide for unique temporary storage (in vector format) for the second and subsequent new blocks generated from a given block vector.

After all blocks in the Main Sub-table have been modified, the contents of the Supplementary Sub-table are transferred to the Main Sub-table, and the cycle is repeated. A detailed description of the selection-modification stage is given by Flowchart 2 of Appendix B.

Outputs

There are two types of outputs generated by NPM, classified as primary and secondary outputs. Primary outputs were illustrated in abbreviated form in Figure 3.1. For the single output option the following information is generated:

1. The binary input matrix,
2. the block vectors corresponding to the blocks forming the partitions,
3. the NSR and EET values for each block in (2),
4. the time-space coordinates of the partitioned matrix, and
5. the partitioned matrix.

For the multiple-output option, the binary input matrix is only printed once for the set of matrices. Items 2 through 5 are printed for each partitioned matrix of the set. In addition, the initial forced zero-level block selection is given for each matrix. This last item provides a means of identification for regenerating a particular matrix of the set.

The secondary printout is optional, and consists of intermediate results of each block selection and subsequent matrix modification, where each selection-modification cycle is referred to as a "level". The

information included in the secondary output is listed as follows;

1. A listing of the IAR Table (i.e., the zero-level block vectors).

2. the NSR and EET values of the zero-level blocks in (1),

3. the block selection for the i-th level,

4. the NSR and EET values of each block, remaining in the matrix prior to modification during the i-th level,

5. the IARB Main Sub-table prior to the modification in the i-th level,

6. the IARB Main Sub-table after i-th level modification,

7. the IARB Supplementary Sub-table after i-th level modification,

8. the Output Vector Sub-table after i-th level modification,

9. the number of zero vectors in the Main Sub-table after i-th level modification, and

10. the number of non-zero vectors in the Supplementary Sub-table after the i-th level modification.

Examples of primary and secondary outputs are illustrated in Figures 3.8 and 3.9 respectively.

Options Available to the User

In its present form, NPM provides the user with several options with regard to the type of output and the objective (i.e., minimization of space requirement or minimization of expected execution time) desired. The options are controlled by switches in the program, and their value must be specified by the user. A summary of these option switches is given below:

	PURPOSE	SWITCH	VALUE	RESPONSE
1.	Block Selection Criterion	IOPTN	1	Blocks selected on basis of maximum NSR
			2	Blocks selected on basis of minimum EET
2.	Single/Multiple Output	JOPTN	0	Single partitioned matrix & associated time-space data

			1	Set of partitioned matrices each with associated time-space data
3.	Secondary Outputs	KSW2	0	Secondary outputs are generated
			1	no secondary outputs generated

A variable, KKNT, is used in conjunction with the JOPTN switch to specify the block vector from which initial forced selections are to begin for multiple output runs. For example, if there are 14 block vectors in the IAR Table and the value of KKNT is 12, three partitioned matrices will be generated by NPM, where the initial block selected for each run is 12, 13, and 14, respectively. For test and experimental purposes, an action weight vector ACTWT and a rule probability vector RLPRB have been incorporated in NPM. Both vectors are initialized by the user by means of DATA statements. Finally, the variable OUTF is used to specify the output device or file.

```
THE BINARY INPUT MATRIX FOLLOWS

 1  1  0  0  0  1  1  0  1  1
 1  0  0  1  1  1  1  1  1  1
 1  1  1  1  1  1  1  1  1  1
 1  1  1  1  1  1  1  1  1  1
 1  1  1  1  0  1  0  1  0  1
 1  1  1  1  1  0  1  0  1  0
 1  1  1  1  0  1  0  1  0  1
 1  1  1  1  1  0  1  0  0  1

THE OUTPUT BLOCK VECTORS FOLLOW

*         .         .         .         *         .         .         .        *****
11000000000000000000000000000010000100010000000000000000000000000 2 3
00000111000000000000000000000001111000000000000000000000000000000 3 4
01110100000000000000000000000000001010100000000000000000000000000 4 3
00111000000000000000000000000001111010101000000000000000000000001 3 7

THE INDIVIDUAL BLOCK VALUES FOLLOW

              NSR             EET

      0.400000E+01    0.000000E+00
      0.900000E+01    0.000000E+00
      0.150000E+02    0.300000E+00
      0.650000E+02    0.700000E+00

THE TIME-SPACE COORDINATES ARE   0.100000E+01    0.930000E+02

          THE PARTITIONED MATRIX FOLLOWS

 1  4  0  0  0  1  9  0 13  1
 1  0  0  5  6  1  6 11  6  1
 2  2  2  2  6  2  6  2  6  2
 2  2  2  2  6  2  6  2  6  2
 2  2  2  2  0  2  0  2  0  2
 3  3  3  3  6  0  6  0  6  0
 3  3  3  3  0  8  0 12  0 14
 3  3  3  3  7  0 10  0  0 15

TERMINATE ALGORITHM
```

FIGURE 3.8

THE IAR TABLE FOLLOWS

```
*       .       .       .       *       .       .       .                 *****
01111111000000000000000000000000100100000000000000000000000000000 7 2
01110101000000000000000000000000010100000000000000000000000000000 5 2
01111010000000000000000000000000010100000000000000000000000000000 5 2
00111111000000000000000000000001111000000000000000000000000000000 6 4
00000011000000000000000000000001111000001000000000000000000000000 2 5
11111010000000000000000000000000010001000000000000000000000001 6 2
11110100000000000000000000000000001010000000000000000000000001 5 2
01110100000000000000000000000000010101000000000000000000000001 4 3
01111010000000000000000000000000010101000000000000000000000001 5 3
11111000000000000000000000000010000100010000000000000000000001 5 3
01111000000000000000000000000010010101010000000000000000000001 4 5
00111000000000000000000000000011110101000000000000000000000001 3 7
11110000000000000000000000000010000110110000000000000000000001 4 5
01110000000000000000000000000010011111110000000000000000000001 3 8
00110000000000000000000000000011111111110000000000000000000001 210
```

THE INITIAL BLOCK VALUES FOLLOW

NSR	EET
0.150000E+02	0.000000E+00
0.900000E+01	0.000000E+00
0.130000E+02	0.000000E+00
0.420000E+02	0.000000E+00
0.800000E+01	0.000000E+00
0.130000E+02	0.200000E+00
0.800000E+01	0.200000E+00
0.150000E+02	0.300000E+00
0.250000E+02	0.300000E+00
0.250000E+02	0.300000E+00
0.470000E+02	0.500000E+00
0.650000E+02	0.700000E+00
0.310000E+02	0.500000E+00
0.480000E+02	0.800000E+00
0.530000E+02	0.100000E+01

FIGURE 3.9

```
THE SELECTED VECTOR FOR LEVEL  0 FOLLOWS

*         .         .         .         *         .         .         .         *****
00111000000000000000000000000000001111010101000000000000000000000000001 3 7

ITS VALUE IS    0.650000E+02

THE BLOCK VALUES FOR LEVEL  0 FOLLOW

      0.150000E+02    0.000000E+00
      0.900000E+01    0.000000E+00
      0.130000E+02    0.000000E+00
      0.420000E+02    0.000000E+00
      0.800000E+01    0.000000E+00
      0.130000E+02    0.200000E+00
      0.800000E+01    0.200000E+00
      0.150000E+02    0.300000E+00
      0.250000E+02    0.300000E+00
      0.250000E+02    0.300000E+00
      0.470000E+02    0.500000E+00
      0.000000E+00    0.000000E+00
      0.310000E+02    0.500000E+00
      0.480000E+02    0.800000E+00
      0.530000E+02    0.100000E+01

THE IARB (MAIN) TABLE FOR LEVEL  0 FOLLOWS

*         .         .         .         *         .         .         .         *****
01111111000000000000000000000000001001000000000000000000000000000001 7 2
01110101000000000000000000000000001010000000000000000000000000000000 5 2
01111010000000000000000000000000001010000000000000000000000000000001 5 2
00111111000000000000000000000001111000000000000000000000000000000001 6 4
00000011000000000000000000000001111000001000000000000000000000000000 2 5
11111010000000000000000000000000001000100000000000000000000000000001 6 2
11110100000000000000000000000000001010000000000000000000000000000000 5 2
01110100000000000000000000000000001010100000000000000000000000000000 4 3
01111010000000000000000000000000001010100000000000000000000000000001 5 3
11111000000000000000000000000000001000100010000000000000000000000001 5 3
01111000000000000000000000000000001001010100000000000000000000000001 4 5
00000000000000000000000000000000000000000000000000000000000000000002 0 0
11110000000000000000000000000000001000011011000000000000000000000001 4 5
01110000000000000000000000000000001001111111000000000000000000000001 3 8
00110000000000000000000000000001111111111100000000000000000000000001 210
```

FIGURE 3.9 (Continued)

```
THE IARB (MAIN) TABLE AFTER LEVEL  0 FOLLOWS

*       .       .       .       *       .       .       .                 *****
00000111000000000000000000000001001000000000000000000000000000001  3  2
01110101000000000000000000000000010100000000000000000000000000000  5  0
00000000000000000000000000000000000000000000000000000000000000002  0  0
00000111000000000000000000000001111000000000000000000000000000001  3  4
00000011000000000000000000000001111000001000000000000000000000000  2  5
11000010000000000000000000000000000010001000000000000000000000001  2  2
11110100000000000000000000000000000010100000000000000000000000000  5  2
01110100000000000000000000000000010101000000000000000000000000000  4  3
00000000000000000000000000000000000000000000000000000000000000002  0  0
11000000000000000000000000000001000010001000000000000000000000001  2  3
00000000000000000000000000000000000000000000000000000000000000002  0  0
00000000000000000000000000000000000000000000000000000000000000002  0  0
11000000000000000000000000000001000011011000000000000000000000001  2  5
00000000000000000000000000000000000000000000000000000000000000002  0  0
00000000000000000000000000000000000000000000000000000000000000002  0  0

THE IARB (SUPP) TABLE AFTER LEVEL  0 FOLLOWS

*       .       .       .       *       .       .       .                 *****
11110000000000000000000000000000000010100000000000000000000000001  4  2
01110000000000000000000000000000001010100000000000000000000000001  3  3
00110000000000000000000000000000010101000000000000000000000000001  2  3

THE OUTPUT VECTORS AFTER LEVEL  0 ARE

*       .       .       .       *       .       .       .                 *****
00111000000000000000000000000001111010101000000000000000000000001  3  7

IARB (MAIN) CONTAINS   6 ZERO VECTORS

IARB (SUPP) CONTAINS   3 NONZERO VECTORS
```

FIGURE 3.9 (Continued)

CHAPTER 4

DISCUSSION OF NPM PERFORMANCE, MULTIPLE-OUTPUT RESULTS, AND CONCLUSTIONS

Comparison of NPM Results With Those of Burnham's BLOCX Algorithm

NPM and BLOCX performance was compared on the basis inherent storage savings for the eight test matrices in Appendix A. In addition, for each test matrix, both a single partitioned matrix and a multiple output set was generated by NPM. The purpose of this step was to determine if an improved solution over the single output would result if alternative zero-level block selections are made. The selection criterion used in NPM for these comparisons was maximization of NSR. In establishing the conditions of the experiment, all actions were considered to be equally weighted (=1) and all rules equally probable. To provide a common basis for comparison of the results, the space savings, reflected in the number of non-programmed actions for each partitioned matrix was recorded. These results are summarized in Table 4.1.

From the results indicated in Table 4.1, it can be seen that NPM produced equivalent results for test matrix 4, and improved results for the seven remaining matrices.

	NUMBER OF NONPROGRAMMED ACTIONS		
TEST MATRIX	BLOCX	NPM (BEST OF SET)	NPM (SINGLE OUTPUT)
1	37	42	39
2	46	52	52
3	25	26	25
4	22	22	22
5	44	55	52
6	71	83	73
7	19	27	27
8	16	17	17

TABLE 4.1

It is especially interesting to note that for half of the test matrices, improvement over the single output results was obtained when the multiple-output set was generated. This confirms our earlier contention, that basing all block selections on a specific criterion does not guarantee the best results. The degree of success obtainable by generating the multiple-output set for a given input matrix, is a function of the zero-level block interrelationships in the input matrix. From our analysis of the test matrices, we suggest the following guidelines for using the multiple-output option of NPM:

1. A complex matrix, where complexity can be measured in terms of the number of over-lapping zero-level blocks in the input matrix will, in general, yield improved results when the multiple output set is generated. This is illustrated by the results obtained for test matrices 5 and 6.

2. A sparse (having few entries) matrix, which has a proprotionately high number of independent zero-level blocks, will not, in general, result in a substantial improvement (if any) when the multiple output option is used. This is illustrated by the results obtained for test matrix 8.

Figures 4.1 and 4.2 illustrate an example of a sparse and a complex matrix, respectively.

Comparison of NPM Results With Those of Penick's Algorithm

The comparison of NPM results with those of the Penick algorithm was not as straightforward as in the case of BLOCX. This is due mainly to an apparent inconsistency of the Penick algorithm in the inclusion of conditional-transfer blocks in the output set. For example, in the case of three of the test matrices (indicated by * in table 4.2), no conditional-transfer blocks were selected,

-THE BINARY INPUT MATRIX FOLLOWS TEST MATRIX 8

```
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0 0 0 1 1 0 0 1 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 1 0 0 0 1 1 0 0 1 0 0 0 0 0 1 0 0 0
0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0
0 0 0 0 1 0 0 0 0 0 1 0 1 1 0 0 1 0 0 0 0 0 1 0
0 0 0 1 0 0 0 0 1 1 0 1 0 1 0 0 0 1 0 0 0 1 1 0
0 0 1 0 0 0 0 1 0 1 1 1 0 0 0 0 1 1 0 0 1 0 0 0
0 1 0 0 0 0 1 1 1 0 0 1 1 0 0 1 0 1 0 1 0 0 1 0
```

-THE IAR TABLE FOLLOWS

```
*       .       .       .       *       .       .       .                 *****
000000010001000000000000000000000000010000000010000000000000000 2 2
000000001010000000000000000000000000000010000010000000000000000 2 2
000000000101000000000000000000000000000010000000000001000000000 2 2
000000000111000000000000000000000000000010000010000000000000000 3 2
000000000011000000000000000000000000100010000010000000000000000 2 3
000000001100000000000000000000000000000010000000010000000000001 2 2
000000000110000000000000000000000000101000001000000000000000001 2 3
000001100000000000000000000000000000000010000010000000000000001 2 2
000001100000000000000000000000000000001001000010000000000000001 2 3
000001100000000000000000000000000000011001000010000000000000001 2 4
000001100000000000000000000000000001000110010000100000000000001 2 5
```

-THE INITIAL BLOCK VALUES FOLLOW

A SPARSE MATRIX

FIGURE 4.1

-THE BINARY INPUT MATRIX FOLLOWS

TEST MATRIX 6

```
0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0
0 0 0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0 1 0
1 1 0 0 0 0 0 1 1 0
1 1 0 0 0 1 0 0 0 0
1 1 0 0 0 1 1 1 0 0
0 0 1 0 0 1 1 1 0 0
0 0 0 1 0 0 0 0 0 0
1 1 1 0 1 1 1 1 1 1
1 1 1 0 1 1 1 1 1 1
1 1 1 0 1 1 1 1 1 1
1 1 1 0 1 1 1 1 1 1
1 1 1 0 1 1 1 1 1 1
1 1 1 0 1 1 1 1 1 1
0 0 0 0 0 1 1 1 1 1
1 1 1 0 1 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1
1 1 1 0 1 0 0 0 0 0
0 1 1 0 1 0 1 0 1 1
1 1 0 0 0 -1 0 0 0 0
1 1 0 0 0 1 0 0 0 0
1 1 0 0 0 1 1 1 0 0
```

-THE IAR TABLE FOLLOWS

```
*       .       .       .       *       .       .       .                 *****
00000000000111111010110000000000010100000000000000000000000000000  9 2
00000000000111111010100000000000000000011000000000000000000000000  9 2
00000000000000000000011100000001100010000000000000000000000000000  3 3
00000000000111111010100000000000000000101100000000000000000000001  9 3
00000000110111111110100000000000000001010000000000000000000000110  2
00000000000111111010110000000000011010000000000000000000000000001  9 3
00000011100111111010100000000000011000000000000000000000000000111  2
00000000110111111110100000000000000011100000000000000000000000110  3
00000000000111111010100000000000111010000000000000000000000000001  8 4
00000000000111111110100000000000000011111000000000000000000000001  8 5
00000000010111111000000000000000000100111000000000000000000000001  7 4
00000000000111111000000000000000111011111110000000000000000000001  6 9
00000001100000000000000000000000011000100000000000000000000000001  2 3
00000110000000000000000000000000010000010000000000000000000000001  2 2
```

-THE INITIAL BLOCK VALUES FOLLOW

A COMPLEX MATRIX

FIGURE 4.2

even though such blocks were available after selection of the direct-transfer blocks were incorporated. The criterion used by NPM for this comparison was minimization of EET, with the provision that conditional-transfer blocks would be allowed after all direct-transfer blocks have been incorporated. The comparison of the results generated by the two algorithms is summarized in Table 4.2.

	PENICK ALGORITHM		NPM	
TEST MATRIX	NSR	EET	NSR	EET
1	26	0.2	31	0.2
2*	21	-	34	0.5
3	18	0.857	18	0.857
4	17	0.364	20	0.364
5*	29	-	36	0.2
6*	34	-	43	0.4

TABLE 4.2

A meaningful comparison between the two techniques is only possible for test matrices 1, 3, and 4, since conditional-transfer blocks were incorporated by both algorithms for these cases. From the results of Table 4.2, we note that the EET values of the three partitioned matrices in question are the same. However, the NSR values of the NPM outputs are higher than the corresponding values

for the Penick algorithm for test matrices 1 and 4. This indicates that the NPM algorithm is more conserative with space requirements while requiring the same increase in expected execution time.

Illustrative Example of Use of the NPM Multiple-Output Option

We indicated in Chapter 3 that NPM was capable of generating a set of partitioned matrices having different time-space requirements. The following discussion will illustrate user requirements for generating a multiple-output set, the type of results generated, and the interpretation of these results. We will use test matrix 1 of Appendix A as the input matrix. For the sake of simplicity we will assume a weight of 1 for each action and a probability of 0.1 for each rule.

The user inputs required to produce a multiple-output set for test matrix 1 are:

1. Set the Single/Multiple Output Switch for multiple output: JOPTN = 1

2. Set the Block Selection Criterion Switch for maximum NSR; IOPTN = 1

3. Set the Secondary Output Switch to inhibit secondary printouts: KSW2 = 1

4. Specify entries of the action weight vector:

   DATA ACTWT/8*1., 24*0./

5. Specify entries of the rule probability vector:

   DATA RLPRB/10*.1, 22*0./

6. Specify file containing the input matrix:

   IDF = Input File Number

7. Specify output device:

   OUTF = Output Device Number

8. Execute NPM

The system will respond with a printout (or other display) of fifteen primary outputs (ref. to Chapter 3) corresponding to the fifteen zero-level block vectors. An illustration of two of these outputs is given in Figures 4.3 and 4.4. We will now discuss the outputs indicated in Figure 4.3.

The first output specifies the initial forced block selection made for that partitioning run. The output block vectors correspond to the blocks forming the partitions of the output matrix, and are followed by the corresponding NSR and EET values for each block vector. The NSR value is proportional (based on the action weights) to the net storage savings inherent in the block represented by the

```
-THE BINARY INPUT MATRIX FOLLOWS

  1  1  0  0  0  1  1  0  1  1
  1  0  0  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  0  1  0  1  0  1
  1  1  1  1  1  0  1  0  1  0
  1  1  1  1  0  1  0  1  0  1
  1  1  1  1  1  0  1  0  0  1

 BLOCK 14 IS THE INITIAL FORCED SELECTION

-THE OUTPUT BLOCK VECTORS FOLLOW

 *         .         .         .         *         .         .         .        *****
 00110000000000000000000000000011000000000000000000000000000000 2 2
 00000101000000000000000000000000010100000000000000000000000000 2 2
 00001010000000000000000000000000001010100000000000000000000000 2 3
 00001111000000000000000000000011110000000000000000000000000001 4 4
 01110000000000000000000000000010011111110000000000000000000001 3 8
-THE INDIVIDUAL BLOCK VALUES FOLLOW

              NSR                EET

         0.200000E+01    0.000000E+00
         0.200000E+01    0.000000E+00
         0.300000E+01    0.300000E+00
         0.120000E+02    0.000000E+00
         0.200000E+02    0.800000E+00
-THE TIME-SPACE COORDINATES ARE   0.110000E+01    0.390000E+02

-        THE PARTITIONED MATRIX FOLLOWS

         1  4  0  0  0  7  9  0 10 12
         2  0  0  2  2  2  2  2  2  2
         2  5  5  2  2  2  2  2  2  2
         2  5  5  2  2  2  2  2  2  2
         3  3  3  3  0  8  0  8  0  8
         3  3  3  3  6  0  6  0 11  0
         3  3  3  3  0  8  0  8  0  8
         3  3  3  3  6  0  6  0  0 13
```

FIGURE 4.3

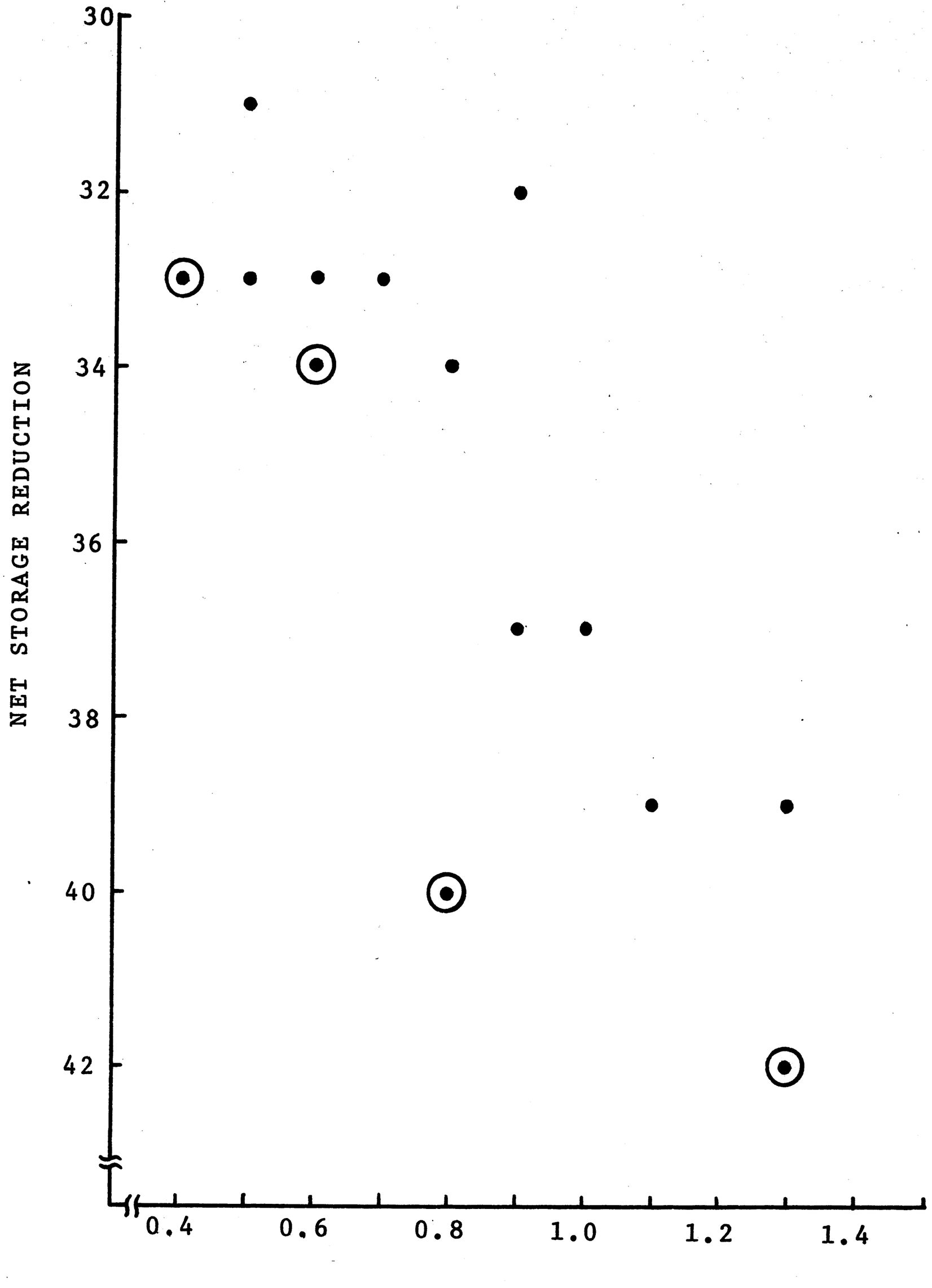
```
 BLOCK 15 IS THE INITIAL FORCED SELECTION

-THE OUTPUT BLOCK VECTORS FOLLOW

 *     .     .     .     *     .     .     .          *****
 0000010100000000000000000000000000010100000000000000000000000000 2 2
 0000010100000000000000000000000000010101000000000000000000000000 2 3
 1100000000000000000000000000000010000110110000000000000000000000 2 5
 0000111100000000000000000000000011110000000000000000000000000001 4 4
 0011000000000000000000000000000111111111100000000000000000000001 210
-THE INDIVIDUAL BLOCK VALUES FOLLOW

              NSR             EET

         0.200000E+01    0.000000E+00
         0.300000E+01    0.300000E+00
         0.800000E+01    0.000000E+00
         0.120000E+02    0.000000E+00
         0.170000E+02    0.100000E+01
-THE TIME-SPACE COORDINATES ARE   0.130000E+01    0.420000E+02

-        THE PARTITIONED MATRIX FOLLOWS

         1  4  0  0  0  1  1  0  1  1
         1  0  0  5  6  1  1  9  1  1
         2  2  2  2  2  2  2  2  2  2
         2  2  2  2  2  2  2  2  2  2
         3  3  3  3  0  8  0  8  0  8
         3  3  3  3  7  0  7  0 10  0
         3  3  3  3  0  8  0  8  0  8
         3  3  3  3  7  0  7  0  0 11
-TERMINATE ALGORITHM
```

FIGURE 4.4

corresponding output vector. The EET value represents the increase in the expected execution time of the matrix, relative to the non-partitioned (rules executed sequentially) matrix, due to the formulation of the block. As discussed in Chapter 1, the only time a non-zero (positive) EET value will occur is in the case of a conditional-transfer block. Thus, the blocks corresponding to the third and fifth block vectors are conditional-transfer blocks, which increase the expected execution time of the matrix 0.3 and 0.8 time units respectively. The time-space coordinates indicate the total EET and NSR for all blocks in the partitioned matrix. Finally, the partitioned matrix itself is printed.

To illustrate how the multiple-output results can be used, we have plotted the generated time-space coordinates in Figure 4.5. Since all coordinates are provided by NPM, such a plot can be generated by using a system plotting routine, if desired. In this plot, the circled entries are the undominated entries of the set. Since the undominated entries represent the most efficient partitionings of the input matrix, it may be desirable to modify NPM to output only these values.

If, as in this example, each action (or decision node) requires one unit of memory for storage and one



INCREASED EXPECTED EXECUTION TIME
TEST MATRIX 1

FIGURE 4.5

unit of time for execution, we can produce a time-space requirement plot based on the following calculations:

1. Calculate the difference between the total space requirement of the matrix before partitioning (i.e., 62 memory units for matrix 1) and each NSR value of Figure 4.5 to obtain the space requirement coordinates

2. Calculate the expected execution time of the matrix before partitioning,

Expected Execution Time of Non-Partitioned Matrix

$$= \sum_{j=1}^{r} p_j \sum_{i=1}^{m} t_i,$$

where, p_j is the rule probability of the j-th rule and t_i is the execution time of the i-th action. Add this value (i.e., 6.2 for matrix 1) to the EET coordinates in Figure 4.5 to obtain the expected execution time requirements.

The corresponding time-space requirement coordinates are then plotted as shown in Figure 4.6. Knowing the time-space requirements of the matrices corresponding to the undominated entries, the analyst will be better equipped to select that partitioned matrix which best meets the constraints of his problem.

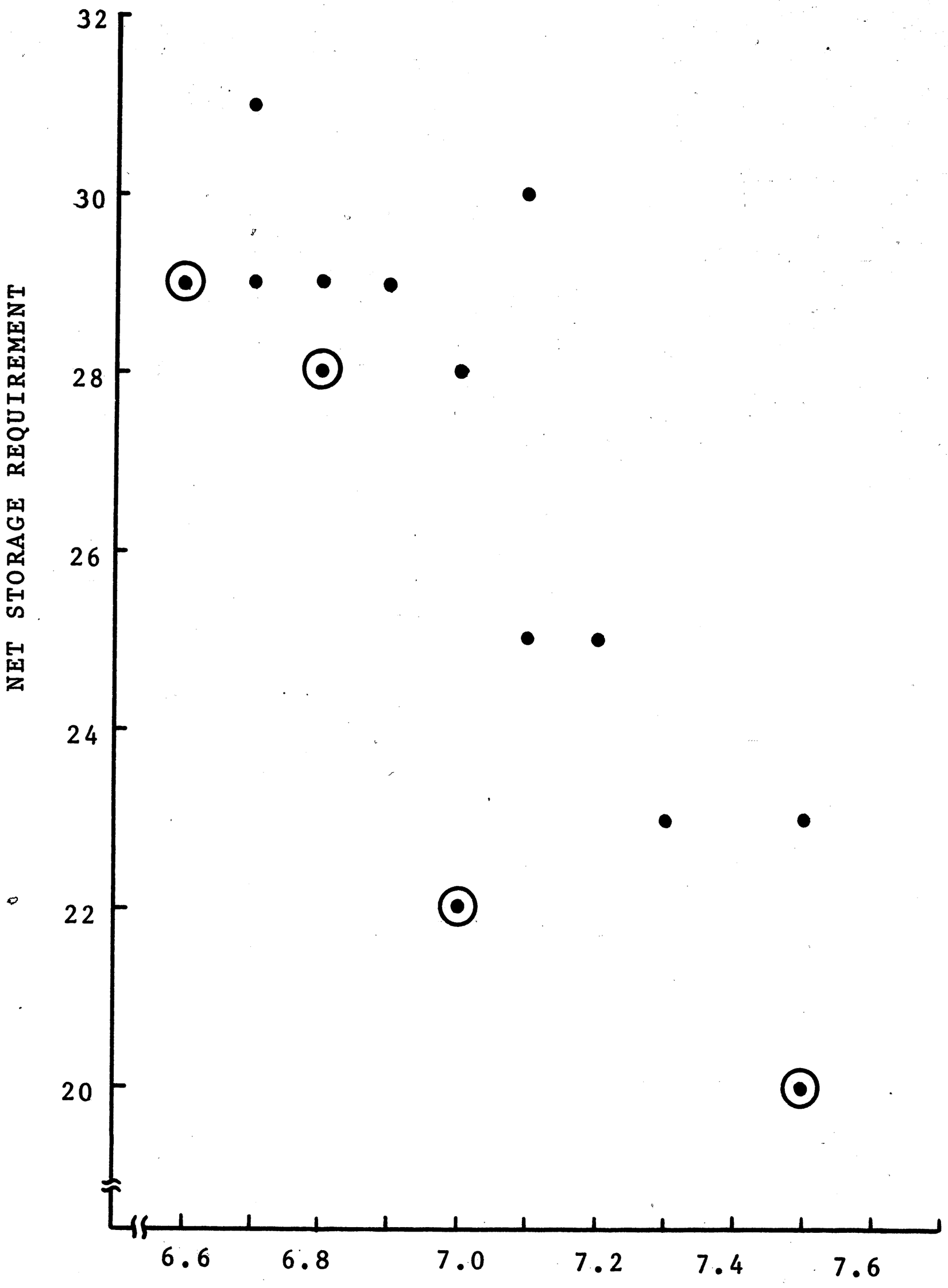

NET STORAGE REQUIREMENT
32
30
28
26
24
22
20
6.6
6.8
7.0
7.2
7.4
7.6
REQUIRED EXPECTED EXECUTION TIME
TEST MATRIX 1

FIGURE 4.6

Conclusions

In this and the previous chapters, we have attempted to illustrate some of the significant advantages offered by the new partitioning module over existing partitioning techniques. We can categorize these improvements in the following areas:

1. Effectiveness: Although comparisons between NPM and existing algorithms were somewhat limited in number, the results tend to indicate the consistency of NPM in producing more effective results than existing algorithms. In the case of maximization of space savings, NPM produced better results than Burnham's BLOCX algorithm for all but one matrix. In the case of minimization of increased expected execution time, NPM produced results identical to those generated by Penick's algorithm. However, NPM resulted in a greater savings in space than the corresponding results of the Penick algorithm. In all of these comparisons, NPM results were at least as good as those of existing algorithms.

2. Flexibility: As we have indicated, NPM is capable of partitioning the action/rule matrix based either on minimization of increased expected execution time or maximization of memory savings, merely by

specifying the desired option. In addition, NPM is not restricted to the use of identical action weights or equal rule probabilities. We feel that removal of these restrictions results in a more realistic approach to partitioning than that offered by existing algorithms.

3. Potential for Application: We feel that the greatest potential for NPM is its use as an analysis tool. We have illustrated how the present capability of the multiple-output option of NPM can be used to provide the analyst with a set of partitioned matrices corresponding to a given input, each having different time and space requirements. This is very useful in determining time space tradeoffs in the case of systems subject to both execution time and storage constraints. With minor modification of the existing algorithm, it will be possible for the user to specify the desired block selection at all selection levels. The information required by the user to make these multi-level selections is already provided by the optional secondary output of NPM.

There is a limitation on the number of rows in the input matrix that NPM will accept. Due to the powers-of-two scheme used to generate the cumulative-sum matrix used for block identification, the maximum row dimension of an input matrix that can be handled by most computers is 32. Some preliminary work has been done on removing this restriction, and an outline of the proposed procedure is presented in Appendix C. This procedure will replace the current block identification stage of NPM without requiring any changes in the selection-modification stage.

APPENDIX A - TEST MATRICES

APPENDIX A

1 1 0 0 0 1 1 0 1 1
1 0 0 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 0 1 0 1 0 1
1 1 1 1 1 0 1 0 1 0
1 1 1 1 0 1 0 1 0 1
1 1 1 1 1 0 1 0 0 1

TEST MATRIX 1

0 0 1 0 0 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
0 1 1 1 1 1 0 1
1 0 1 1 1 1 1 0
0 1 1 1 1 1 0 1
1 0 1 1 1 1 1 0
0 1 1 1 1 1 0 1
1 1 1 1 0 0 0 0

TEST MATRIX 2

1 0 0 0 0 0 0
1 1 0 0 0 0 0
1 1 1 0 0 0 0
1 1 1 1 0 0 0
1 1 1 1 1 0 0
1 1 1 1 1 1 0
1 1 1 1 1 1 1
1 1 1 1 1 1 0
1 1 1 1 1 0 0
1 1 1 1 0 0 0
1 1 1 0 0 0 0
1 1 0 0 0 0 0
1 0 0 0 0 0 0

TEST MATRIX 3

1 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 1 0 0 1
0 0 0 0 1 0 0 1 0 0 1
0 0 1 0 0 1 0 0 0 0 0
0 0 1 1 1 1 0 0 1 0 0
0 0 0 1 0 0 1 0 0 1 0
0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1
0 0 1 1 0 1 1 0 0 0 0
0 0 1 1 1 1 1 1 1 1 1
1 0 1 1 1 1 1 1 1 1 1
0 1 0 0 0 0 0 0 0 0 0

TEST MATRIX 4

0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0
0 0 0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0 1 0
1 1 0 0 0 0 0 1 1 0
1 1 0 0 0 1 0 0 0 0
1 1 0 0 0 1 1 1 0 0
0 0 1 0 0 1 1 1 0 0
0 0 0 1 0 0 0 0 0 0
1 1 1 0 1 1 1 0 1 1
1 1 1 0 1 1 1 0 0 1
1 1 1 0 1 1 1 1 0 0
1 1 1 0 1 1 1 0 1 0
1 1 1 0 1 1 1 0 0 0
1 1 1 0 1 1 1 1 0 1
0 0 0 0 0 1 0 0 1 1
1 1 1 0 1 0 1 0 0 0
0 0 0 0 0 1 0 1 0 0
1 1 1 0 1 0 0 0 0 0
1 1 1 0 1 0 1 0 1 1
1 1 0 0 0 0 1 1 1 1
1 1 0 0 0 1 0 0 0 0
1 1 0 0 0 1 1 1 0 0

TEST MATRIX 5

0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0
0 0 0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0 1 0
1 1 0 0 0 0 0 1 1 0
1 1 0 0 0 1 0 0 0 0
1 1 0 0 0 1 1 1 0 0
0 0 1 0 0 1 1 1 0 0
0 0 0 1 0 0 0 0 0 0
1 1 1 0 1 1 1 1 1 1
1 1 1 0 1 1 1 1 1 1
1 1 1 0 1 1 1 1 1 1
1 1 1 0 1 1 1 1 1 1
1 1 1 0 1 1 1 1 1 1
1 1 1 0 1 1 1 1 1 1
0 0 0 0 0 1 1 1 1 1
1 1 1 0 1 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1
1 1 1 0 1 0 0 0 0 0
0 1 1 0 1 0 1 0 1 1
1 1 0 0 0 1 0 0 0 0
1 1 0 0 0 1 0 0 0 0
1 1 0 0 0 1 1 1 0 0

TEST MATRIX 6

0 1 1 1 0
0 0 0 0 1
1 1 1 1 0
0 0 0 0 1
0 0 0 1 0
0 0 1 0 1
0 1 0 1 0
0 1 1 1 1
0 1 1 1 1
0 1 1 1 1
0 0 0 1 1
0 0 0 1 1
0 0 0 1 1
0 1 1 1 1
0 1 1 1 1
0 1 1 1 1
0 1 1 1 1
0 1 1 0 0
0 1 1 0 0

TEST MATRIX 7

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0

0 0 0 0 0 0 1 0 0 0 1 1 0 0 1 0 0 0 0 0 1 0 0

0 0 0 0 0 0 1 0 0 0 1 1 0 0 1 0 0 0 0 0 1 0 0

0 0 0 0 0 1 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0

0 0 0 0 1 0 0 0 0 0 1 0 1 1 0 0 1 0 0 0 0 0 1

0 0 0 1 0 0 0 0 1 1 0 1 0 1 0 0 0 1 0 0 0 1 1

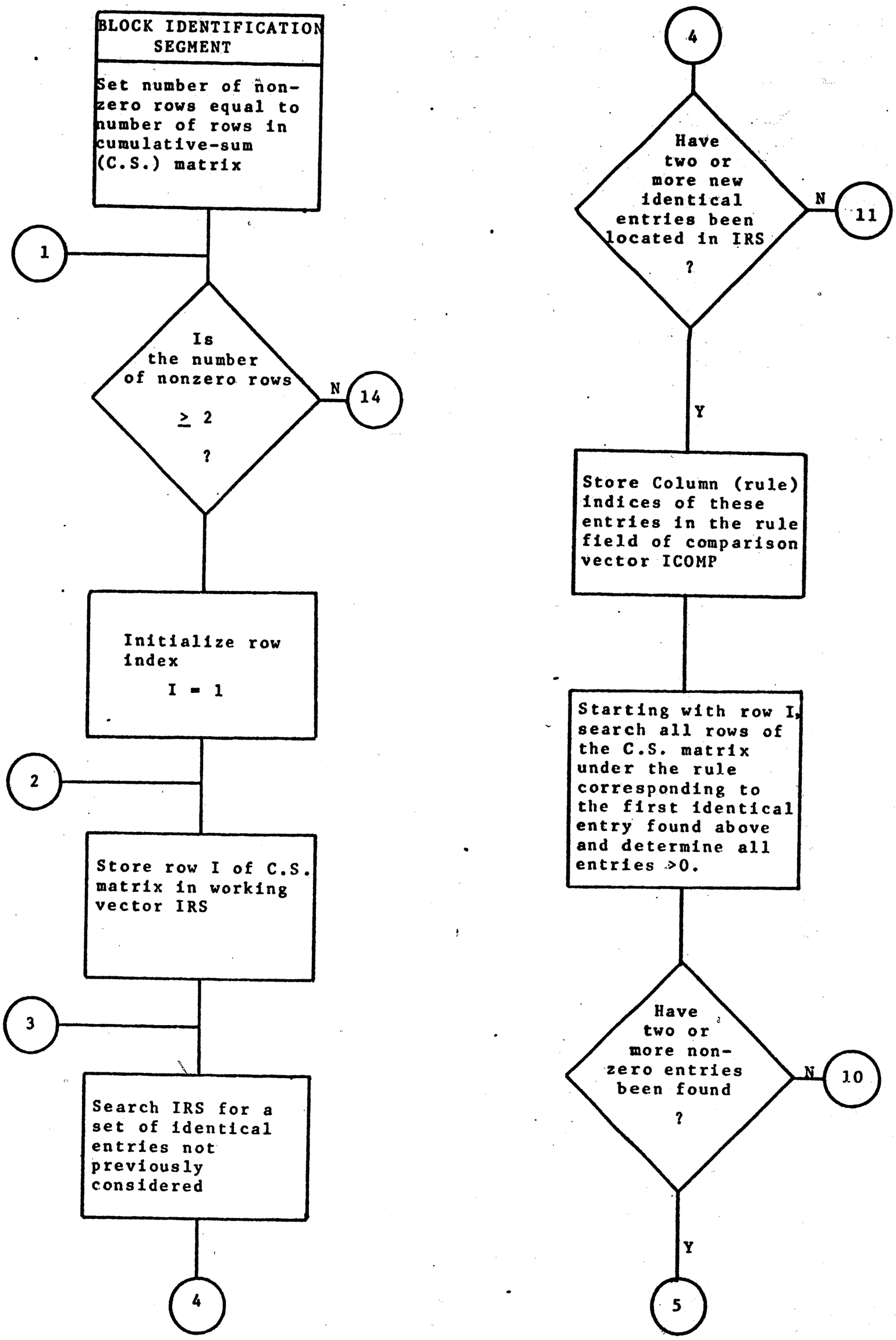
0 0 1 0 0 0 0 1 0 1 1 1 0 0 0 0 1 1 0 0 1 0 0

0 1 0 0 0 0 1 1 1 0 0 1 1 0 0 1 0 1 0 1 0 0 1

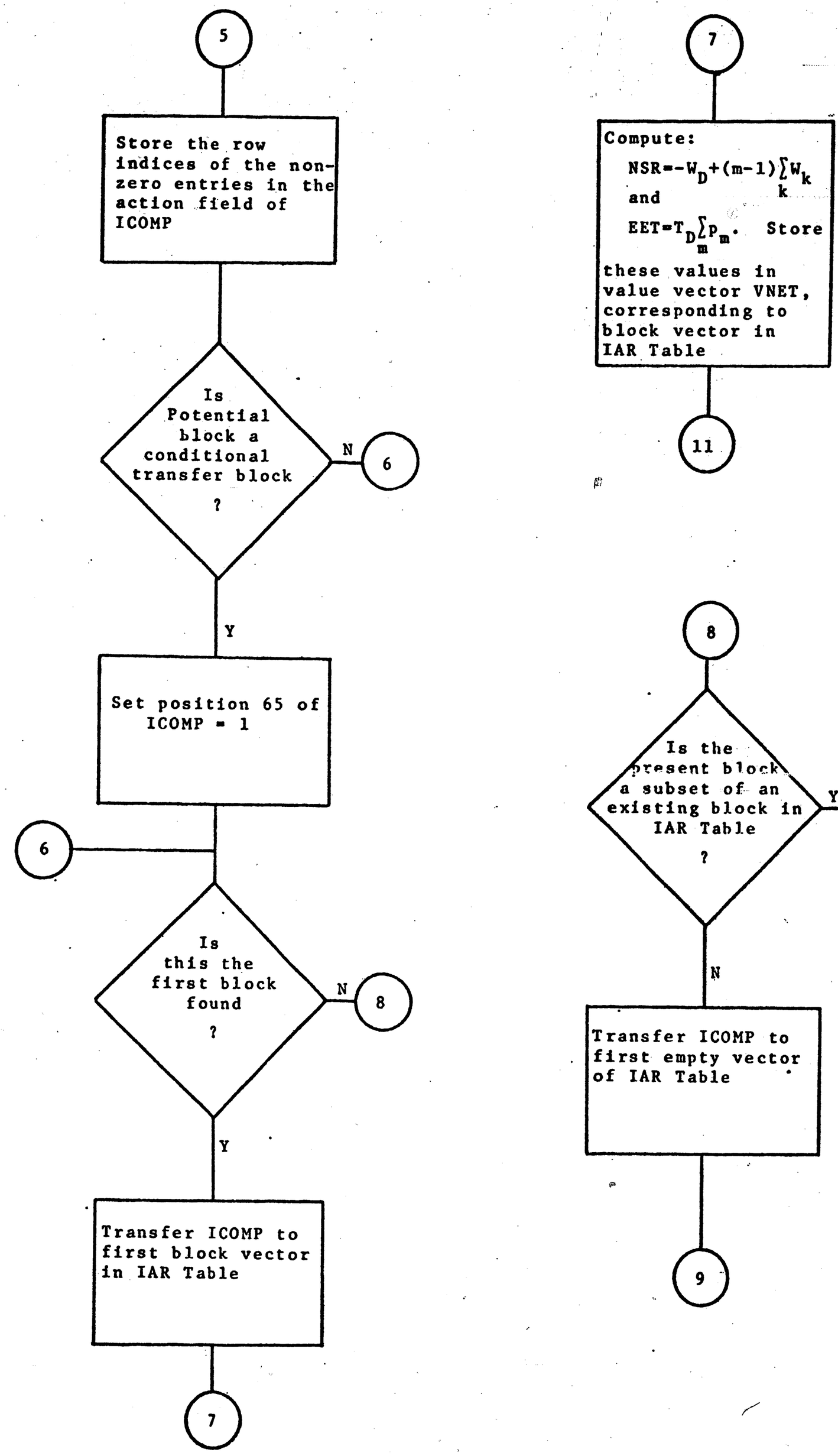
TEST MATRIX 8

APPENDIX B - PROGRAM FLOWCHARTS

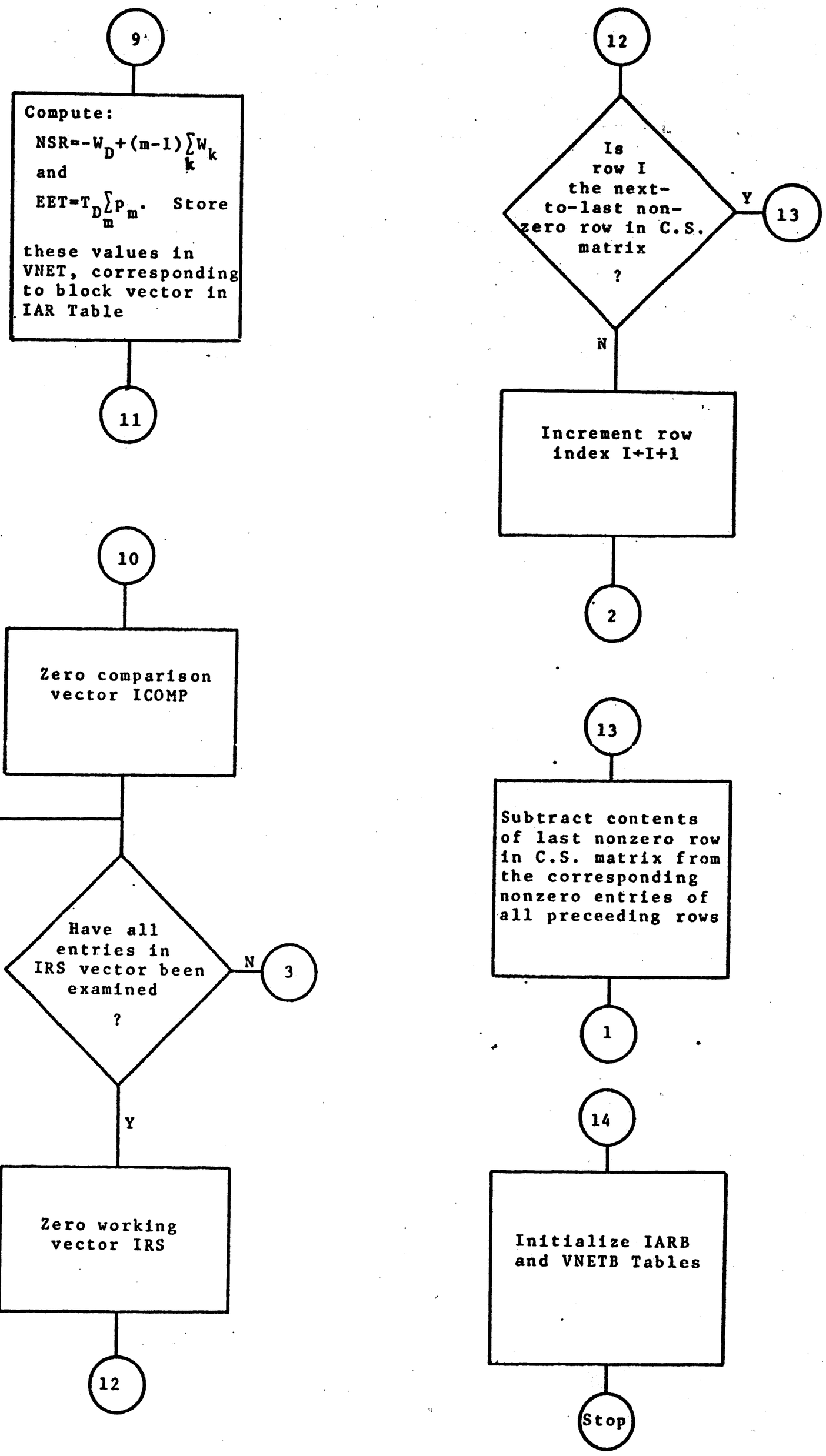
APPENDIX B



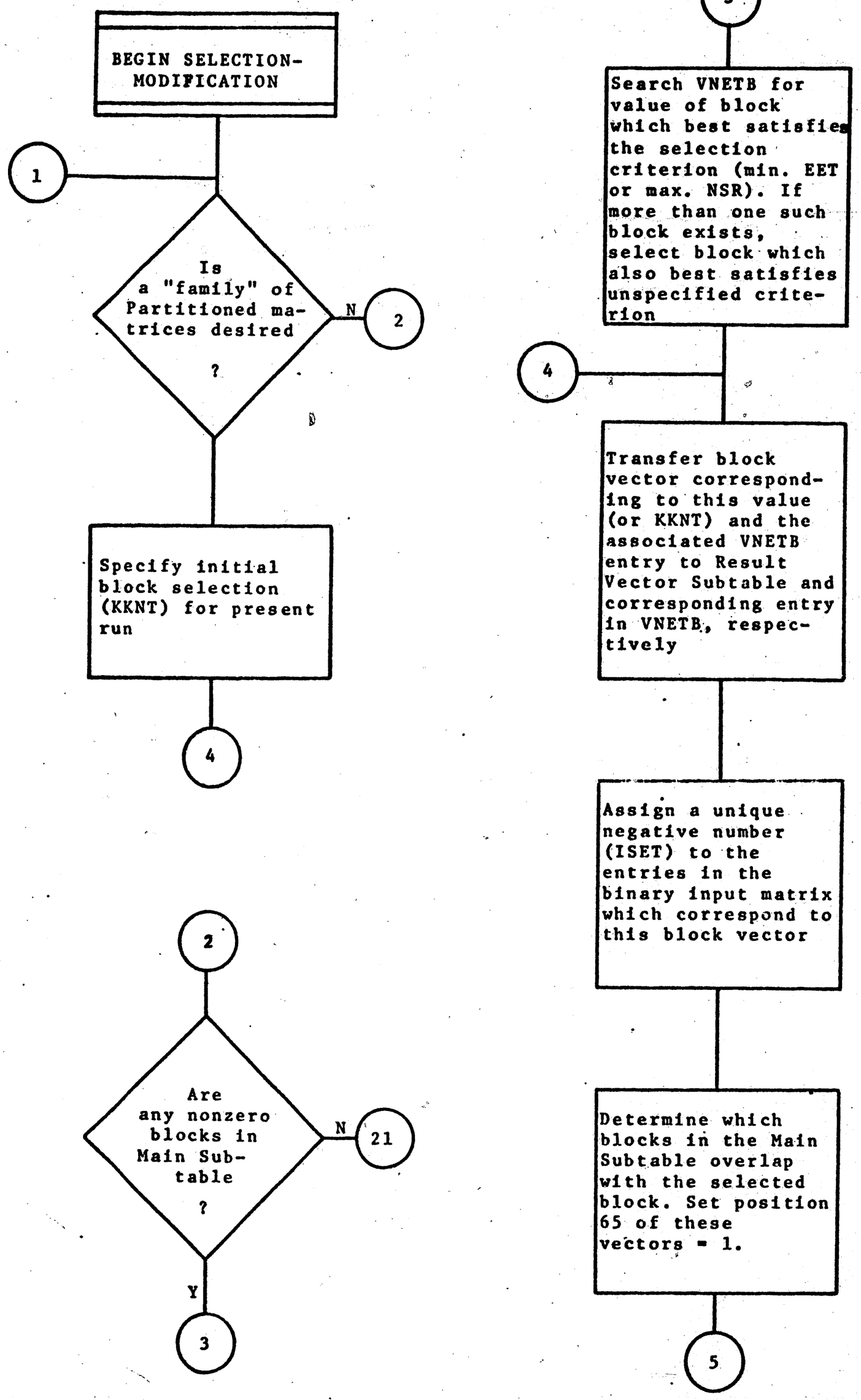
FLOWCHART 1



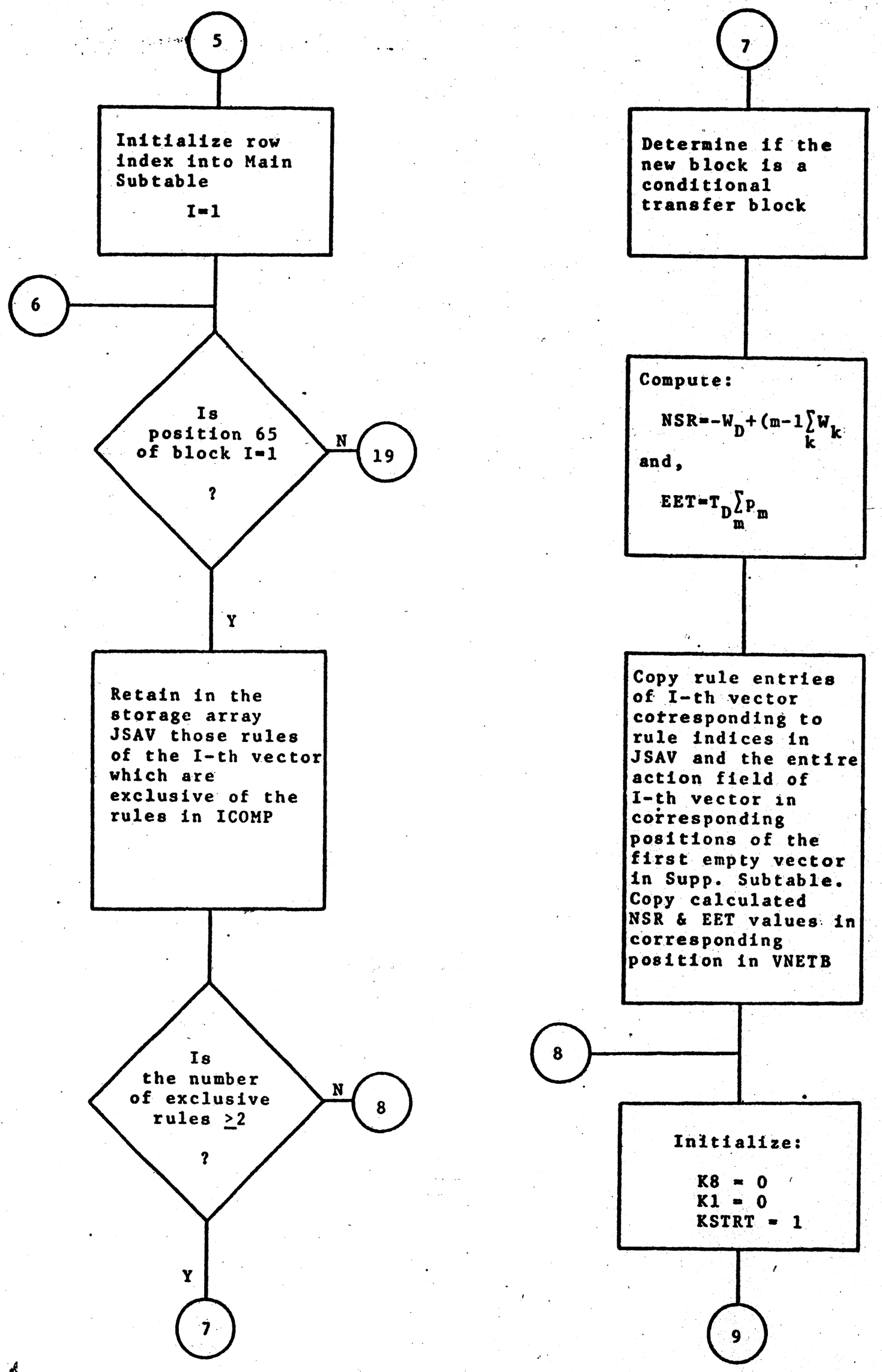
FLOWCHART 1 (Continued)



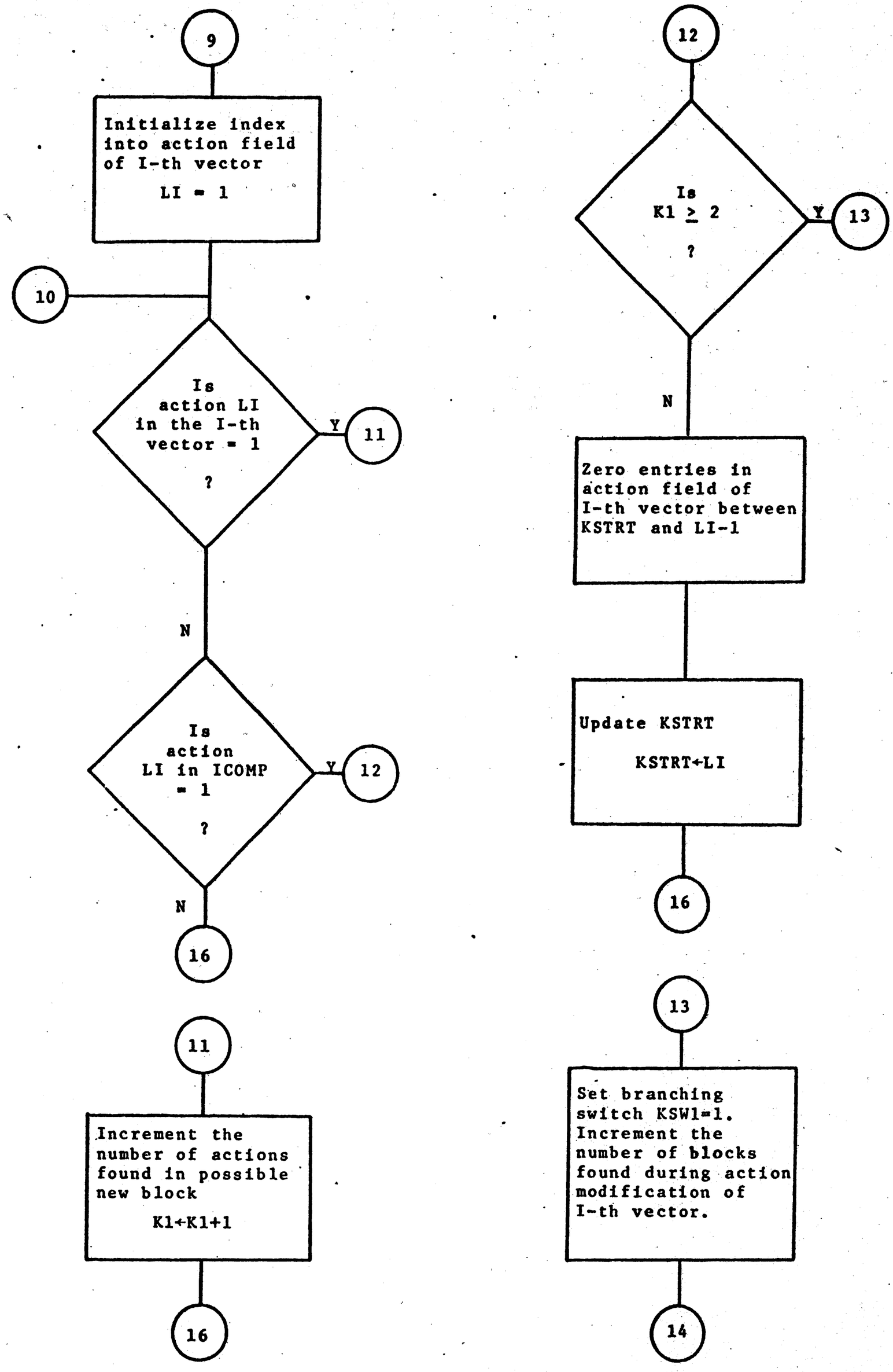
FLOWCHART 1 (Continued)


BEGIN SELECTION-MODIFICATION
1
Is a "family" of Partitioned matrices desired ?
N
2
Specify initial block selection (KKNT) for present run
4
2
Are any nonzero blocks in Main Subtable ?
N
21
Y
3
3
Search VNETB for value of block which best satisfies the selection criterion (min. EET or max. NSR). If more than one such block exists, select block which also best satisfies unspecified criterion
4
Transfer block vector corresponding to this value (or KKNT) and the associated VNETB entry to Result Vector Subtable and corresponding entry in VNETB, respectively
Assign a unique negative number (ISET) to the entries in the binary input matrix which correspond to this block vector
Determine which blocks in the Main Subtable overlap with the selected block. Set position 65 of these vectors = 1.
5

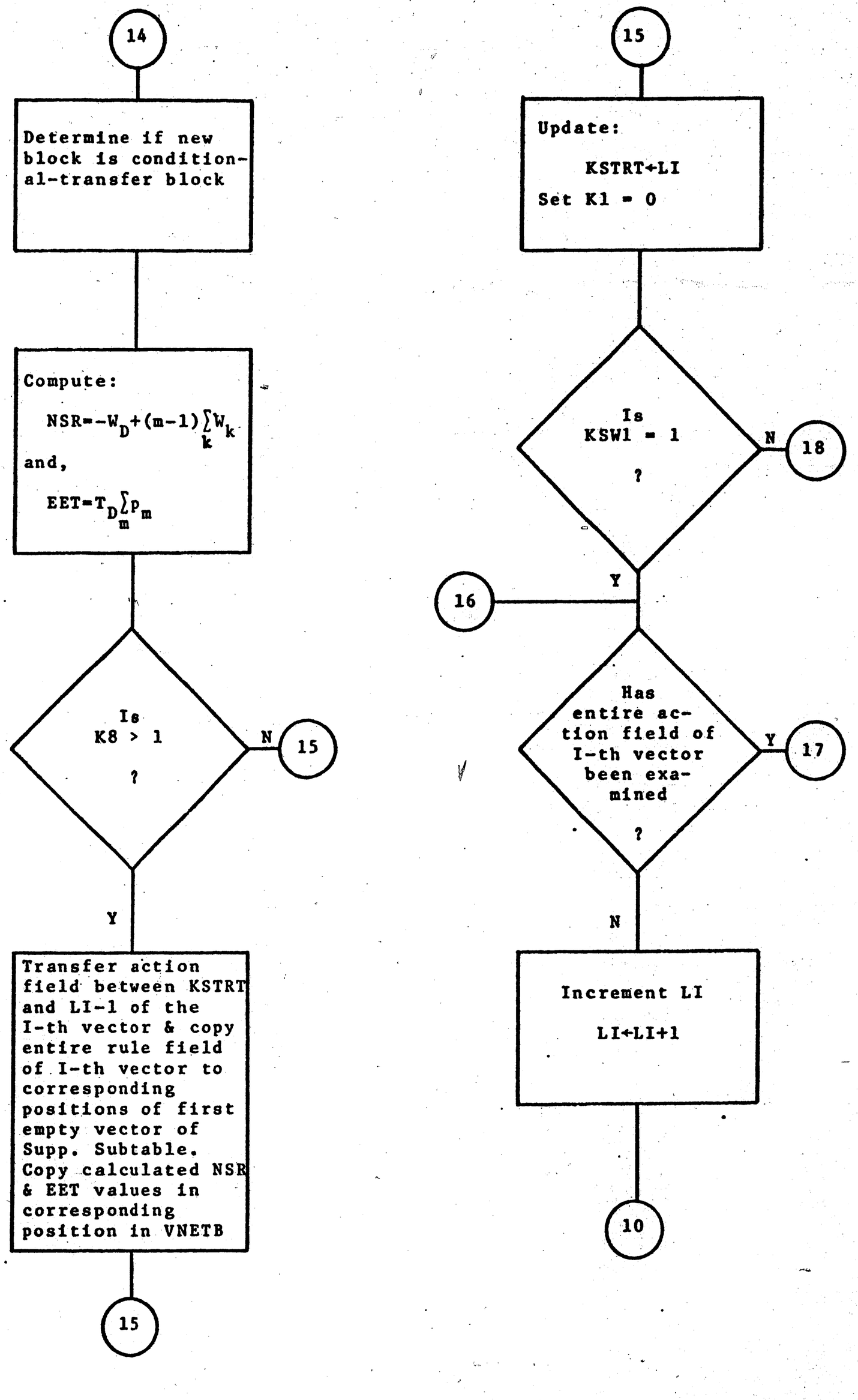
FLOWCHART 2



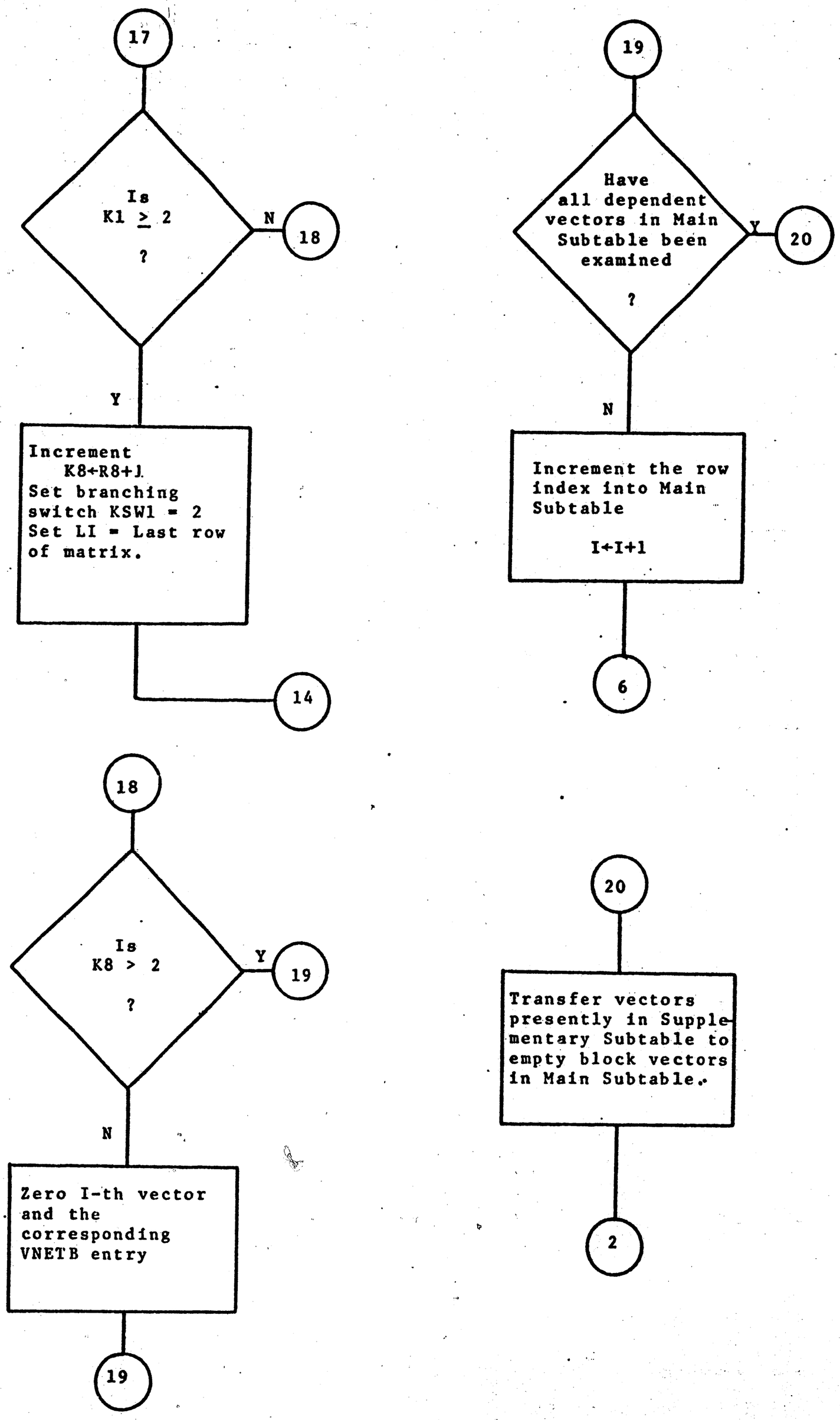
FLOWCHART 2 (continued)



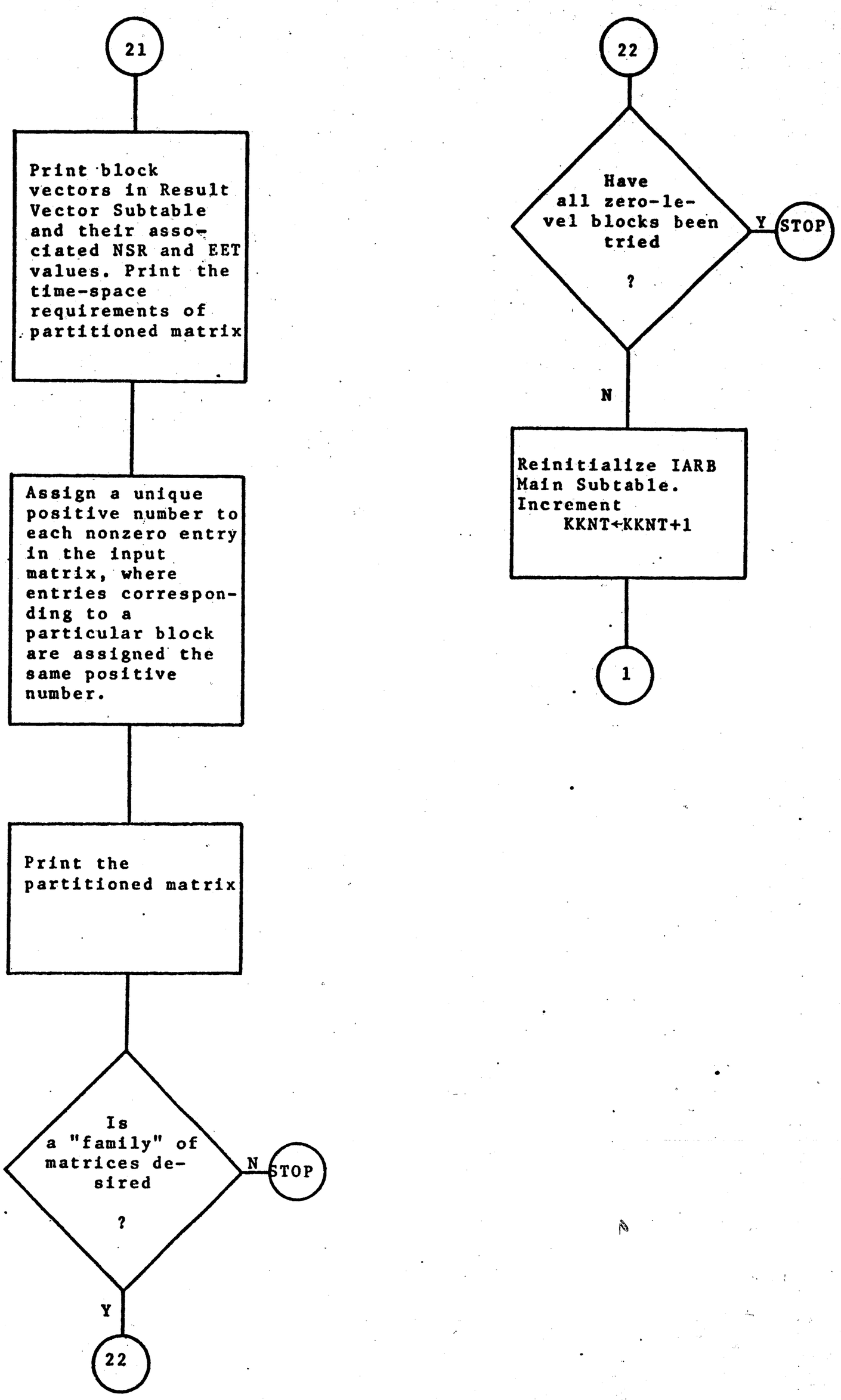
FLOWCHART 2 (continued)



FLOWCHART 2 (continued)



FLOWCHART 2 (continued)



FLOWCHART 2 (continued)

APPENDIX C - PROPOSED BLOCK IDENTIFICATION STAGE

APPENDIX C

At the present time, NPM will only accpet matrices having a maximum of 32 rows. This is a consequence of the powers-of-two scheme used to produce the cumulative-sum matrix required for block identification. We propose a possible approach to eliminating the powers-of-two cumulative sum matrix approach to block identification. The results using the suggested approach will allow the use of the present selection-modification stage of NPM.

Consider the binary input matrix of Figure 5.1

	R1	R2	R3	R4
A1	1	1	1	1
A2	0	1	1	1
A3	1	0	1	1
A4	1	1	0	1
A5	1	1	1	0

FIGURE 5.1

STEP 1 Form the exclusive-or of all rule combinations as shown in Figure 5.2, where the column headings specify the rules of the original matrix involved in the exclusive-or.

(1,2)	(1,3)	(1,4)	(2,3)	(2,4)	(3,4)
0	0	0	0	0	0
1	1	1	0	0	0
1	0	0	1	1	0
0	1	0	1	0	1
0	0	1	0	1	1

FIGURE 5.2

STEP 2 Determine the runs of zeros in each column of Figure 5.2 which contain at least two entries whose value in the original matrix is 1. The runs which meet this criterion are shown in Figure 5.3.

ROW	(1,2)	(1,3)	(1,4)	(2,3)	(2,4)	(3,4)
1	0	0	0	0	0	0
2	1	1	1	0	0	0
3	1	0	0	1	1	0
4	0	1	0	1	0	1
5	0	0	1	0	1	1

FIGURE 5.3

STEP 3 From Figure 5.3, form the intersections of all runs, and retain those results which have more than two entries. There is only one such result, namely

ROW	(2,3,4)
1	0
2	0

Retain all runs which are not proper subsets of larger runs. In this case there are three such runs, namely

ROW	(3,4)
1	0
2	0
3	0

ROW	(1,4)
3	0
4	0

ROW	(1,2)
4	0
5	0

These four runs represent the all of the blocks in the input matrix. Since the row and column indices are known, we can store this information in block vector format.

BIBLIOGRAPHY

1. Beizer, B., "The Architecture and Engineering of Digital Computer Complexes, Volume 1", Plenum Press, 1971.

2. Burnham, M. J., Private Communication, Western Electric Co., Princeton, N. J., Concerning the Partitioning of the Action Part of a Limited-Entry Decision Table, May 1973.

3. Dingee, W. L., "Pragmatic Problems of Decision Table Optimization", M. S. Thesis, Department of Industrial Engineering, Lehigh University, Bethlehem, Pennsylvania, 1973.

4. Dixon, P., "Decision Tables and Their Application", Computers and Automation, April 1964.

5. Grad, B., "Tabular Form in Decision Logic", Datamation, July 1961.

6. Montalband, M., "Tables, Flowcharts and Program Logic", IBM Systems Journal, September 1962.

7. Penick, L. C., Private Communication, Western Electric Co., Princeton, N.J., Concerning the Partitioning of the Action Part of a Limited Entry Decision Table, February 1972.

8. Rabin, J., "Conversion of Limited-Entry Decision Tables into Optimal Decision Trees: Fundamental Concepts", Sigplan Notices, September 1971.

9. Reinwald, L. T., and Soland, R. M., "Conversion of Limited Entry Decision Tables to Optimal Computer Programs I: Minimum Average Processing Time", Journal of the ACM, July 1966.

10. _____, "Part II: Minimum Storage Requirements", Journal of the ACM, October 1967.

11. Yasu, Tashio, "Some Aspects of Decision Table Conversion Techniques", Sigplan Notices, September 1971.

12. _____, "Conversion of Decision Tables into Decision Trees", Ph.D. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, February, 1972.

VITA

Personal History

Name: Rolf G. Pinckert
Date of Birth: April 27, 1943
Place of Birth: Plauen, Germany
Parent: Martha T. Vinson
Wife: Sandra M. Pinckert
Children: Stephanie and Valerie Pinckert

Educational Background

Menard Memorial High School
Alexandria, Louisiana (1958-1961)

Louisiana Polytechnic Institute
BSEE - 1969

Lehigh University
MSIE (Candidate) - 1974

Honors

Tau Beta Pi
Eta Kappa Nu (President of local chapter)

Professional Experience

United States Navy
Electronics Technician
Petty Officer 2nd Class
1961-1965

Western Electric Co.
Southwestern Region
Ballwin, Missouri
Engineer 1969-1971
Planning Engineer 1972

Lehigh Masters Program
Western Electric Corporate
Education Center 1972-1974