1973

Pragmatic problems of decision table optimization

Wallace L. Dingee
Lehigh University

PRAGMATIC PROBLEMS
OF
DECISION TABLE OPTIMIZATION

by

Wallace Lee Dingee, Jr.

A Thesis

Presented to the Graduate Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Industrial Engineering

Lehigh University
1973

CERTIFICATE OF APPROVAL

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

April 10, 1973
Date

W. A. Smith, Jr.
Professor in Charge

O. P. Lovell
Chairman of the Department of
Industrial Engineering

ACKNOWLEDGEMENTS

The author would like to express his sincere appreciation to Dr. William A. Smith Jr., of the Industrial Engineering Department of Lehigh University, for his interest, advice and guidance during the preparation of this thesis. Special thanks are due Mr. Jonas Rabin, of the Western Electric Engineering Research Center Staff, for his technical support and helpful suggestions. Ms. Clarissa King also deserves special thanks for her patience and expertise in the typing of this manuscript.

Finally the author wishes to express his gratitude to his wife, Joy, for her help in proofreading and her encouragement during the course of this effort.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

The conversion of a decision table into an optimal decision tree is an integral part of most decision table compilers. In batch processing the emphasis is on maximizing throughput. For computing systems with multiprogramming capabilities (most common today), the objective shifts to minimizing storage requirements. Likewise, in a transaction-oriented (real-time) system, the objective is to minimize the response time to transactions. Heuristic approaches to optimization have proven to be more practical than optimal seeking methods that use mathematical programming techniques. This thesis establishes some fundamental concepts of a heuristic approach to the conversion of limited entry decision tables into decision trees and proposes algorithms for the latter two objectives.

The model used is basically that of a complete decision tree. Partial decision trees are created by the elimination of condition tests from the complete tree. The relationship between dash (immaterial) entries in a decision table and eliminated sub-trees is developed. The algorithms include a method for predetermining which conditions must be tested for procedural purposes, even though they are immaterial to a particular rule from a logical point of view. Probabilities are associated with the rules and test times with the conditions when the objective is to minimize response time. Examples illustrating the performance of the proposed algorithms are presented and compared to approaches discussed in the literature.

CHAPTER I

INTRODUCTION AND BACKGROUND

A. Description of Decision Table

Before any discussion of decision tables or decision table optimization can begin, it is advisable to define the principal characteristics involved. A decision table may be described as an orderly, non-procedural 2-dimensional representation of the logic necessary to solve a problem. We could say that the creation of a decision table forces the identification of exactly what set of conditions must be satisfied before a given set of actions can be performed and facilitates the identification of elements which have no influence on a set of actions.

B. History of Decision Tables

Practically everyone experiences some contact with tables in everyday life and makes decisions based on these tables. While these tables may assist us in making decisions, they rarely codify decision rules based on specific conditions and resultant actions. While tables in general have been around for quite a long time, decision tables are relatively new. In 1957 General Electric began an "Integrated Systems Project" to study manufacturing processes and the role of computers. Because of the complex logic structures involved, a method of expression called "decision structure tables" was developed. At approximately the same time the Sutherland Company was developing decision tables as an aid to systems analysis. By 1959 Hunt Foods had also begun to use decision tables, and in May of

that year the conference on Data Systems Languages (CODASYL) convened and dedicated one of its committees to the objective of developing a machine independent, systems oriented language. This committee began to study decision tables and after two years produced the decision table language, DETAB-X. General Electric presented their work on decision tables in 1960 at the Eastern Joint Computer Conference and then spent the next two years developing decision table processors. During this time IBM, Rand, Boeing and the Insurance Company of North America also worked on processors. The period from 1962 until 1965 was for the most part inactive. But in June of 1965 the Special Interest Group for Programming Languages (SIGPLAN) appointed a working group to develop a decision table preprocessor. The preprocessor was written in COBOL, accepted tables coded in COBOL, and produced COBOL source code. Even though its implementation was widespread, its inefficient conversion algorithm and lack of maintenance led to its eventual disappearance. Since 1960 the decision table processors written have been COBOL oriented. The exceptions are: (1) IBM's System/360 Decision Logic Translator that processes decision tables coded in FORTRAN and (2) ERCTRAN developed at the Western Electric Engineering Research Center which was written in FORTRAN. According to Pollack[13], the history of decision table processors can be viewed as consisting of four eras:

1. The era of initial development, 1957-1960.
2. The first era of preprocessors, 1961-1962.
3. The era of silence, 1963-1965.
4. The second era of preprocessors, 1966-present.

C. Types of Tables and Forms

Decision tables are generally viewed as falling into one of two categories, limited entry and extended entry. A combination of the two being considered mixed entry. The limited entry table consists of conditions, rules, and action sets where each rule is described by indicating that a condition must be satisfied, need not be satisfied or is immaterial. These three states are designated by the following symbols: 1, 0, or -. A rule's action set consists of the particular action or actions and their sequence which are to be taken at execution time for the particular rule.

The precise actions to be taken and their sequence are variable from one rule to another. A simple limited entry table is shown in Figure 1.1. Since there are three conditions involved in the table in Figure 1.1, there are 2^3 possible unique orderings or simple rules as candidates for rules in the table. The dash in condition two in Rule 1 implies that both of the simple rules $\begin{matrix}1\\0\\1\end{matrix}$ and $\begin{matrix}1\\1\\0\end{matrix}$ make up rule 1. That is, they both result in the execution of action 1. A similar observation can be made for rules two and three in the table. From our observations we see that six simple rules are represented by the rules 1, 2 and 3 in the table. This leaves 8 - 6 = 2 simple

	Rule 1	Rule 2	Rule 3	ELSE Rule
Condition 1	1	-	0	
Condition 2	-	0	1	
Condition 3	0	1	-	
Actions 1	X		X	
2		X	X	
3				X

FIGURE 1.1

rules to be thrown into a catch all called the ELSE rule, with its corresponding ELSE action set. In this case the ELSE action set consists of action three. Also in Figure 1.1 we see that no simple rule appears in more than one rule in the table, thus we say that the rules in the table are mutually exclusive implying that there are no redundancies or contradictions. At this point, to avoid confusion, we will define a transaction to be the state of the condition variable(s) for all conditions at execution time.

Even though this thesis will be concerned with limited entry decision tables only, the form of an extended entry table is presented in Figure 1.2 for contrast. An extended entry table actually contains part of the condition in the rule entries. This is necessary when, in the construction of the table, one chooses not to represent the state of the conditions in the rule entries as strictly yes or no answers. An obvious alternate approach is to break the complex conditions up into a series of simpler conditions resulting in yes or no answers. This is the common procedure for converting extended entry tables to limited entry tables.

	R1	R2	R3	ELSE Rule
C1	>6	=6	<3	
C2	=7	=8	<7	
C3	=2	<2	>2	
	A1	A2	A3	AE

FIGURE 1.2

D. Some Advantages and Disadvantages

The arguments for and against decision tables could develop into a lengthy discussion. However, a nicely compiled list has been presented by J. Ellis[2]. Among some of the advantages listed are that decision tables:

1. "Force the analyst to make a complete and accurate statement of the problem logic."
2. "Ease the modularity design of programs, since actions are broken into small packages and handled separately from condition testing."
3. "Provide better communication between the manager, the analyst and the programmer."
4. "Are relatively easy to create, to change, and to read; it is no great problem to extend the conditions, rules, and/or actions in a table."
5. "Provide an excellent form of documentation since all of the sources of a specific action can be seen."
6. "Make it relatively easier to separate application logic from computer procedural considerations."

Statements 1, 2, 4, 5, and 6 together provide a basis for statement 3. Ellis also states one disadvantage of decision tables. "When creating decision tables there can exist a strong temptation for the systems analyst to over specify." That is, the nature of decision tables encourages the identification of all logical possibilities and these must be combined or designated in the ELSE rule, if

possible, in order to avoid excessively large programs.

E. The Optimization Criteria

Since the purpose of a decision table translator is to translate decision tables into computer programs, an effort should be made to produce an optimal translation subject to some criteria. Thus, the term decision table optimization is used and refers to the object program produced at its execution time. Two widely accepted criteria, which are also used in this thesis, are the approaches of (1) minimizing storage requirements and (2) minimizing the average response (run) time of the object program. The concept of minimizing a program's storage requirements helps to maximize overall throughput in a batch environment. This statement is made under the assumption that as programs require smaller amounts of storage, more programs can be run. There are three things that play a part in either a program's size or its response time when it is generated by a decision table translator. These are: (1) the way in which the action sets are coded, (2) the order or sequence in which the conditions are tested, and (3) the efficiency of the compiler used to create the object program. But if we assume that the action sets are already coded and that the compiler has already been chosen, then the problem of optimization lies in our treatment of the conditions. Two common methods of translation are those using rule mask techniques and those using decision trees. Essentially the rule mask techniques create a binary data vector from a transaction by testing each condition in the table and then compare the data vector to the rules to determine

which action set to execute. The decision tree approach does not necessarily require that all conditions be tested for every rule. Usually, only the conditions pertinent to a particular rule need to be tested. The resulting tree can be called a partial tree. Thus there appears to be an advantage of the decision tree approach over rule mask techniques when the objective is to minimize response time. The minimization of storage depends on the size and structure of the table itself. Using rule mask techniques, the amount of storage is a function of the number of conditions and the number of rules. The storage required when a decision tree is used is a function not only of the number of conditions and rules but also the number and position of dashes in the table.

F. Why the Heuristic Approach to Optimization

According to Rabin[19], the published algorithms for the conversion of decision tables into decision trees can be divided into two classes: (1) complex and time consuming algorithms using mathematical programming techniques and (2) simpler heuristic algorithms which give "good" but not necessarily optimal trees. The optimal tree is defined as the one with a minimal number of non-terminal nodes or the one whose expected response time to a transaction is minimal according to the optimization criteria. Optimal seeking methods such as the branch and bound algorithms proposed by Reinwald and Soland[19,20] require an enormous amount of computation time. This is easily understood when it is realized that there are

$$n \cdot (n-1)^{2} \cdot (n-2)^{2^{2}} \cdot (n-3)^{2^{3}} \cdot \cdot \cdot \cdot n - (n-2)^{2^{n-2}}$$

$$= \prod_{i=0}^{n-2} (n-i)^{2^{i}}$$

possible complete trees for n conditions. Even though the existence of dashes and an ELSE rule lowers the number of complete trees that must be examined, the result is still quite large. For instance, with only six conditions the Reinwald-Soland algorithm for the minimum response time criteria would calculate 65,636 bounds at the fifth level in the tree alone, not to mention the time spent in possible backtracking due to the branching process. Because of possible changes in the structure of a decision table which would necessitate the recomputation of a decision tree, such "guaranteed optimal" procedures are of doubtful practical value. Therefore the need for heuristic algorithms providing "good" but not necessarily optimal trees is apparent.

G. Goal of the Thesis

Heuristic algorithms for the minimum storage case using decision trees, as described in the literature, do not provide any justification or theoretical background. With the exception of an algorithm by Press, they fail to utilize the ELSE rule in any of their calculations. Press, on the other hand, does not utilize the dashes in any of his calculations. Heuristic algorithms for the

minimum response time case do not allow for an ELSE rule but require that the simple rules comprising the ELSE rule be a part of the table. These algorithms also assume that each condition takes the same amount of time to test. Therefore, it will be the goal of this thesis to develop the fundamental concepts of a complete tree approach to decision table optimization and present two algorithms, one for each optimization criteria, minimum storage and minimum response time. The concepts will include: (1) the relationship between dashes in the decision table and non-terminal node elimination in the decision tree, (2) a "look-ahead" strategy to foresee non-terminal node eliminations due to the ELSE rule, (3) a way to detect dash splitting, (4) the interdependence of the ELSE rule and dash eliminations of node sets and (5) the assigning of variable condition testing times for minimum response time considerations.

CHAPTER II

INVESTIGATION OF LITERATURE

A. Rule Mask Techniques

One of the first methods for converting decision tables into computer programs was a rule mask technique proposed by Kirk [8]. It is nothing more than a scanning process with certain masking provisions to handle the don't care on dash entries in the table. As an example consider the table in Figure 2.1:

	R1	R2	R3	R4	ELSE
C1	1	0	1	0	
C2	0	-	1	0	
C3	1	1	-	0	
	A1	A2	A3	A4	AE

FIGURE 2.1

Kirk himself made no mention of the ELSE rule but for completeness it can be easily added. The first step is to replace the dash entries with zeros creating a new table. At the same time a masking matrix consisting of all ones, except for zeros corresponding to the original dashes, is formed. For the example in Figure 2.1 we would have:

New Table

	R1	R2	R3	R4	ELSE
C1	1	0	1	0	
C2	0	0	1	0	
C3	1	1	0	0	
	A1	A2	A3	A4	AE

Masking Matrix $\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}$

By testing the conditions each transaction is converted to a binary data vector which is masked (logical AND) by the first column in the masking matrix and compared with the first rule. If the two vectors are not equal, the data vector is masked by the second column in the masking matrix, compared with the second rule, and so on. The procedure terminates when either a match occurs, causing execution of a particular action set, or the rules are exhausted, causing execution of the ELSE rule action set. If a transaction results in the data vector, $\begin{bmatrix}0\\1\\1\end{bmatrix}$, then the logical product or AND function of the data vector and the first column of the masking matrix is:

$$\begin{bmatrix}0\\1\\1\end{bmatrix} \times \begin{bmatrix}1\\1\\1\end{bmatrix} = \begin{bmatrix}0\\1\\1\end{bmatrix}$$

. But $\begin{bmatrix}0\\1\\1\end{bmatrix}$ does not equal Rule 1 or $\begin{bmatrix}1\\0\\1\end{bmatrix}$. Therefore, we move to the second column of the masking matrix. This results in $\begin{bmatrix}0\\1\\1\end{bmatrix} \times \begin{bmatrix}1\\0\\1\end{bmatrix} = \begin{bmatrix}0\\0\\1\end{bmatrix}$ = Rule 2 (R2).

Thus the transaction is identified as corresponding to Rule 2 and the appropriate action set (A2) can be executed.

An obvious advantage of the technique as with all rule mask techniques, is that each condition is tested only once in determining the data vector. Since the mask table is binary and requires a relatively small amount of storage, the total storage required is proportional to the number of conditions. However, because of the scanning procedure, the technique can produce undesirable run times in contrast to a decision tree approach where the approximate run time is just that time involved in testing the pertinent conditions

of a rule in a predefined sequence (path through the tree).

A similar rule mask technique proposed by Muthukrishnan and Rajaraman[10] provided for the detection of ambiquities at execution time. They arqued that even though two rules were not mutually exclusive (i.e., they do not differ in at least one condition) they should still be allowed in a table. For example, if two rules appeared in a table as follows:

Ri	Rj
1	-
-	0
0	0
A1	Aj

, then a "real" ambiguity exists only when $\begin{bmatrix}1\\0\\0\end{bmatrix}$ is <u>logically</u> possible. And supposedly, one only knows at execution time whether or not the transaction $\begin{bmatrix}1\\0\\0\end{bmatrix}$ is logically possible or not; thus ruling out a decision tree approach.

At this point their technique would identify the ambiguity. However, the logic of their discussion in a <u>practical</u> sense is not as convincing. First of all, if the creator of the table is not sure that $\begin{bmatrix}1\\0\\0\end{bmatrix}$ is a logical possibility, it would seem reasonable that he would not want to create a table and use a technique that offers the possibility of bombing out at execution time. Secondly, if a <u>mistake</u> has been made in the <u>creation</u> of the table, it would seem advantageous to determine the ambiguity at <u>translation time</u> (using decision trees) rather than at execution time. And thirdly, if the originator of the table truly knows that $\begin{bmatrix}1\\0\\0\end{bmatrix}$ is a logical impossibility, then there is no need to use Muthukrishnan and Rajaraman's technique in the first place unless one is trying to handle an <u>error condition</u>. This could be equally provided for by a path corresponding

to $\begin{bmatrix}1\\0\\0\end{bmatrix}$ in a decision tree. However, to see the mechanics of their method, consider the following table:

	R1	R2	R3	ELSE
C1	1	-	0	
C2	-	0	1	
C3	0	0	-	
	A1	A2	A3	AE

Let Y $\triangleright \begin{bmatrix}1\\0\end{bmatrix}$, N $\triangleright \begin{bmatrix}0\\1\end{bmatrix}$, - $\triangleright \begin{bmatrix}1\\1\end{bmatrix}$

then the table becomes:

	R1	R2	R3	ELSE
C1	1	1	0	
	0	1	1	
C2	1	0	1	
	1	1	0	
C3	0	0	1	
	1	1	1	
	A1	A2	A3	AE

Figure 2.2

If a particular transaction results in the binary data vector, $\begin{bmatrix}0\\1\\1\end{bmatrix}$ it then becomes $\begin{bmatrix}0\\1\\1\\0\\1\\0\end{bmatrix}$.

If the modified table and transaction are combined as,

$$\begin{matrix} \\ * \\ * \\ \\ * \\ \end{matrix}\begin{bmatrix}1&1&0\\0&1&1\\1&0&1\\1&1&0\\0&0&1\\1&1&1\end{bmatrix} \quad \begin{bmatrix}0\\1\\1\\0\\1\\0\end{bmatrix}$$

*rows corresponding to ones in the transaction.

and the rows of the table corresponding to the ones in the transaction are written separately;

then the result is: $S = \begin{bmatrix}0&1&1\\1&0&1\\0&0&1\end{bmatrix}$. A column with all ones here

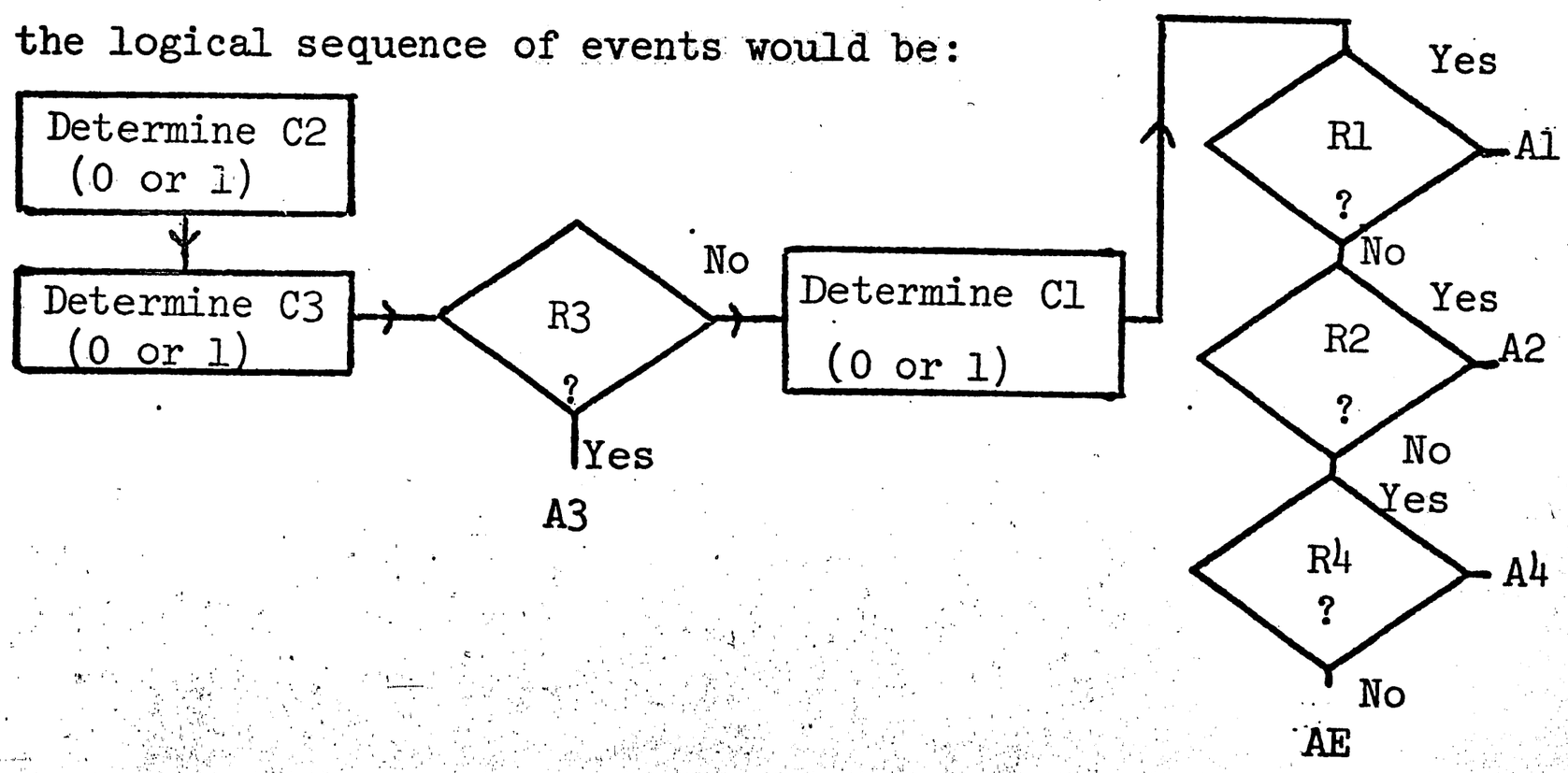
implies a match between transaction and rule. If you logically AND the rows you have R = [0 0 1]. Therefore, our transaction corresponds to Rule 3 and the appropriate action set is executed. If the vector R has more than one non zero element, then there is an ambiguity. (i.e., more than one column in S with all ones).

The inherent tendency of the rule mask techniques to produce programs with longer run times than those produced by decision trees was investigated by King[6]. Instead of evaluating all conditions in a transaction in order to create the binary data vector used for comparison with the rules as proposed by Kirk; King proposed only testing the pertinent conditions of a rule in creating the data vector. Consider the following table:

	R1	R2	R3	R4	ELSE
C1	1	1	-	0	
C2	1	-	0	1	
C3	0	1	0	-	
	A1	A2	A3	A4	AE

FIGURE 2.3

If the rules are tested in the order of R3, R1, R2 and R4, then the logical sequence of events would be:



Thus, we see that the order in which the rules and conditions are tested can determine run time, given that we know the relative frequencies of the rules and the times associated with testing the conditions. King developed four strategies for finding conditions and rule testing sequences. The four strategies usually yield different results and the one that produce a testing sequence with the lowest total run time is selected for use. The first strategy tests the conditions in decreasing order of the sum of the rule frequencies, say $\sum f_i$, where the condition is pertinent, That is, the sum of the rule frequencies where the condition does not have a dash entry. The second strategy tests the conditions in descending order of the $\sum f_i$ divided by the time it takes to test the condition. This gives a measure of the pertinence of a condition relative to its test time. The third strategy tests the rules in descending order of frequency, evaluating conditions (either o or 1) only when they become necessary for testing the rule. The fourth and last strategy used by King tests the rules in descending order of f_i divided by the sum of the times for testing the pertinent conditions in a rule. This is based on the assumption that rules with lower frequencies may still be good first choices if their evaluation times are also low. King did not discuss or comment on which strategy was the best but instead suggested trying all four and choosing the best. The technique can be shown to offer a marked saving in run time as opposed to Kirk's method, especially on large tables.

B. Decision Tree Techniques

Montalbano[10] seems to have provided the initial thinking for the "network technique" or "decision tree" algorithms. His most sophisticated algorithm is called the "modified delayed rule method". The underlying heuristic involved is nothing more than always discriminating on the condition that splits the table into two sub-tables that are equal or close to equal in size. Since dashes indicate the combination of simple rules, the number of simple rules represented by a single rule in the table is calculated as $2^{\text{No. Dashes}}$. Montalbano avoided testing conditions with no dashes as long as possible. To illustrate, consider the table in Figure 2.4:

	R1	R2	R3	R4	ELSE
C1	1	1	0	0	
C2	0	-	0	1	
C3	0	1	-	-	
C4	9	0	1	-	
	A1	A2	A3	A4	AE
	(1)	(2)	(3)	(4)	Rule weight (No. Simple Rules)

Count Matrix

	0	1	-
C1	6	3	0
C2	3	4	2
C3	1	2	6
C4	3	2	4

Figure 2.4

The count matrix gives the number of simple rules corresponding to each condition's zeros, ones or dashes (i.e., the sum of the rule weights for the particular type of entry). For this example the algorithm would choose C1 as the root node in a decision tree, since the entry in the count matrix for the dash is zero (i.e., a condition row without dashes is always preferred).

Montalbano's algorithm touched on the concept of considering the complete tree, representing all simple rules, when he calculated the rule weights. However, the ELSE rule, which also contains simple rules, was not considered. Furthermore, as will be seen in Chapter III of this thesis, it is the simultaneous evaluation of the effect of dashes in the table and the use of the ELSE rule that produces the program (dicision tree) requiring the least amount of storage (relative to other heuristics in the literature).

Ironically, Press[18] in his algorithm tended to take advantage of the ELSE rule while ignoring the dashes. He realized that when a condition row was all ones or all zeros, the opposite branch in the decision tree immediately called for execution of the ELSE action set. This actually causes an elimination of half of the <u>complete tree</u>, as will be seen in Chapter III. Press' numerical calculations were presented with practically no logical justification whatsoever. It will be shown however, that it is possible to calculate the number of non-terminal nodes (conditions) that can be eliminated from the complete tree due to the ELSE rule; and that these eliminations become successively smaller as you "look ahead" down a single path in the complete tree. Before illustrating Press' algorithm, its decision rules are presented:

(1) Always choose a condition to test that has all zeros or all ones, if possible.

(2) Choose a single condition that has no dashes, if possible.

(3) For the zero's group and one's group of each condition, look for complementing entries in the other conditions and count the number of rules involved. Whenever there is a complimenting condition, choose the condition that maximizes the count.

In Figure 2.5 C1's zero's group has no complementing entries. The one's group, however, has two complementing entries with

	R1	R2	R3	R4	ELSE
C1	1	0	1	0	C1's count = 0 + 2 + 2 = 4
C2	1	-	0	1	C2's count = 0 + 0 + 2 = 2
C3	0	1	0	0	C3's count = 1 + 0 + 1 = 2
C4	1	1	1	0	C4's count = 1 + 1 + 1 = 3
	A1	A2	A3	A4	AE

Figure 2.5

two rules involved in each. Thus C1's count is 0 + 2 + 2 = 4. For this example, C1 would be the root node in the decision tree (i.e., the first condition tested).

Pollack[14] proposed two algorithms for converting decision tables to decision trees; one for minimum storage considerations and the other for minimum response (run) time considerations. He made calculations based on the dashes in a table, but did not examine the fact that the ELSE rule provides a way of creating a tree smaller than the complete tree; even when a condition row is not all ones or zeros. In other words, he did not try to look ahead as was attempted by Press. Because of these points and the physical

insignificance of his calculations regarding the dashes, his algorithm can be shown to fail; especially when there is a large number of simple rules associated with the ELSE rule. Pollack, himself, laid no logical foundation for his algorithms other than the fact that they were similar to Montabano's. He personally stated[14]: "I now describe two algorithms, but offer no proof that they accomplish their objectives. Hopefully, others will develop the necessary proofs or offer counter-examples to prove that the algorithms fail."

The two calculations required for each table or sub-table are the column count and the dash count. A third calculation called delta is made to break ties in the dash count. The column count for each rule is found by raising the number two to the number of dashes in the column (i.e., the number of simple rules represented by the single rule in the table). The dash count for each condition is the sum of the column counts associated with the dashes, if any, in the condition. Delta, for each condition, is the absolute value of the difference between the sum of the column counts for the one's group (excluding dashes) and the zero's group. The algorithm is illustrated in Figure 2.6.

Pollack's algorithm for the minimum response (run) time case is a simple extension of his minimum storage algorithm. The dash counts become weighted dash counts (WDC) by multiplying the column counts by the rule frequencies before summing them. Since the ELSE rule is not considered in the calculations, the algorithm becomes more and more ineffective as the frequency of the ELSE increases.

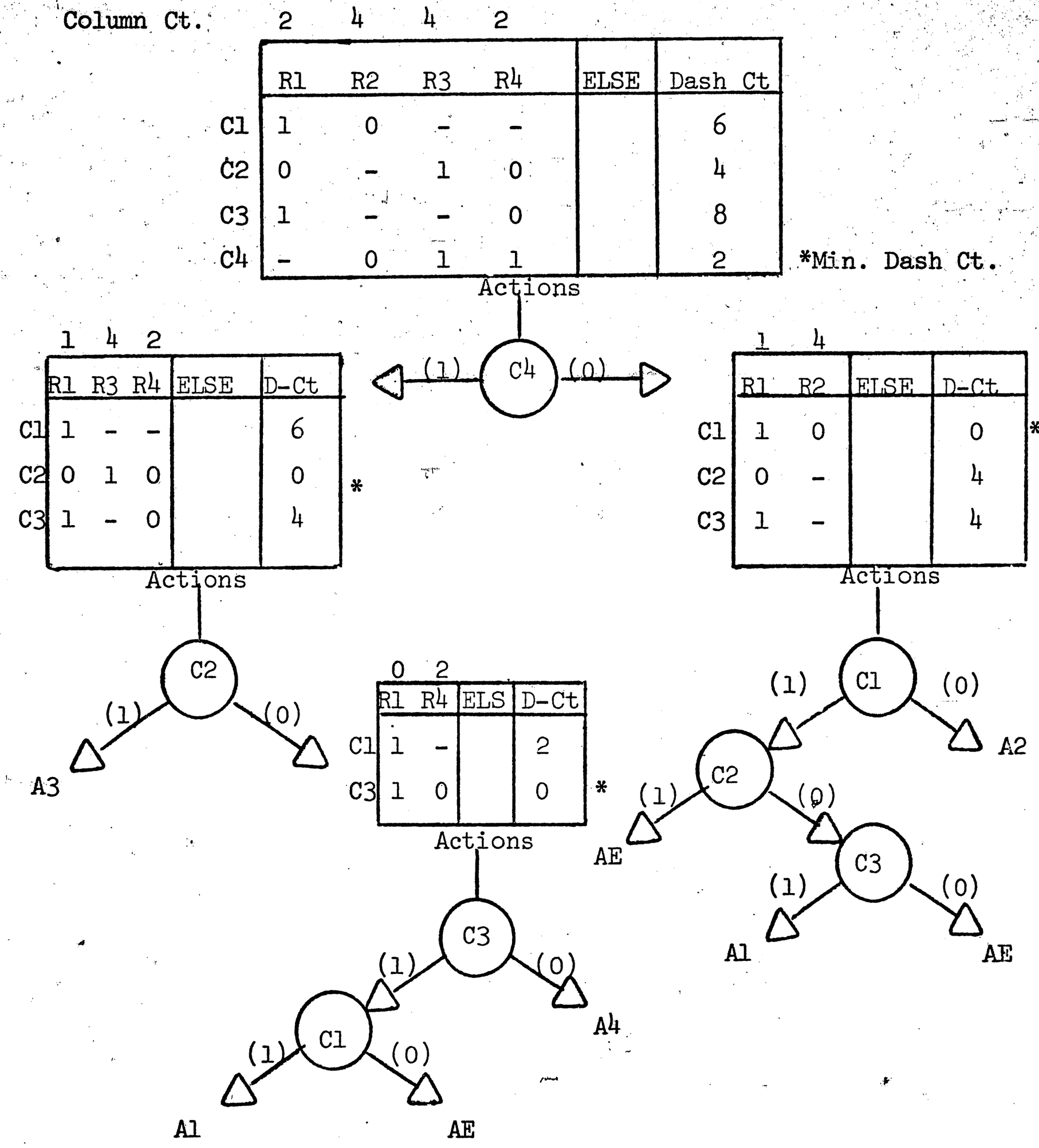

Column Ct. 2 4 4 2
R1 R2 R3 R4 ELSE Dash Ct
C1 1 0 - - 6
C2 0 - 1 0 4
C3 1 - - 0 8
C4 - 0 1 1 2
*Min. Dash Ct.
Actions
C4
(1)
(0)
1 4 2
R1 R3 R4 ELSE D-Ct
C1 1 - - 6
C2 0 1 0 0 *
C3 1 - 0 4
Actions
C2
(1)
(0)
A3
1 4
R1 R2 ELSE D-Ct
C1 1 0 0 *
C2 0 - 4
C3 1 - 4
Actions
C1
(1)
(0)
A2
C2
(1)
(0)
AE
C3
(1)
(0)
A1
AE
0 2
R1 R4 ELS D-Ct
C1 1 - 2
C3 1 0 0 *
Actions
C3
(1)
(0)
A4
C1
(1)
(0)
A1
AE

FIGURE 2.6

Furthermore, all conditions are regarded as having the same test time. An example table is translated into a decision tree in Figure 2.7 to illustrate the algorithm.

The concept of information theory was brought into use in pro ducing decision trees by Shwayder[22]. The algorithm requires relative frequencies (probabilities) for each rule in the decision table. If a rule has dash entries, then the probability of the rule is the sum of the probabilities of the simple rules comprising the rule. Shwayder did not consider the costs (times) associated with testing the conditions. Furthermore, the ELSE rule has to be broken into all of its simple rules if the algorithm is to be applied.

According to Abramson[1], the "entropy" or average amount of information per source symbol of a zero memory binary source is:

$$P_0 \log \frac{1}{P_0} + P_1 \log \frac{1}{P_1}.$$

Shwayder defines P_1 to be the conditional probability of a condition being a one and P_0 the conditional probability of a condition being a zero, given that they are not dashes. Then by letting P equal the probability of a condition not being a dash, he computes the entropy of a condition (row) with the expression: $E = P(P_1 \log \frac{1}{P_1} + P_0 \log \frac{1}{P_0})$.

The condition with the highest entrophy is always chosen. Since $P_1 \log \frac{1}{P_1} + P_0 \log \frac{1}{P_0}$ has its maximum value, which is one, when

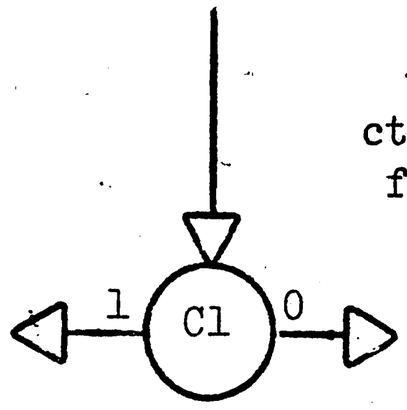
$P_1 = P_0 = .5$; it is then seen that the value of E is also maximized if P = 1. We can say that E, for a condition, is really a weighted

(ct) Column Count = 2 4 4 2

(f)Rule Freq. = 50 25 10 10 5

	R1	R2	R3	R4	ELSE	WDC
C1	1	0	-	-		60
C2	0	-	1	0		100
C3	1	-	-	0		140
C4	-	0	1	1		100
	A1	A2	A3	A4	AE	

*minimum weighted dash count



ct= 2 2 1
f =50 10 10

	R1	R2	R3	ELSE	WDC
C2	0	1	0		0
C3	1	-	0		20
C4	-	1	1		100

ct = 4 2 1
f = 25 10 10

	R2	R3	R4	ELSE	WDC
C2	-	1	0		100
C3	-	-	0		120
C4	0	1	1		0

*

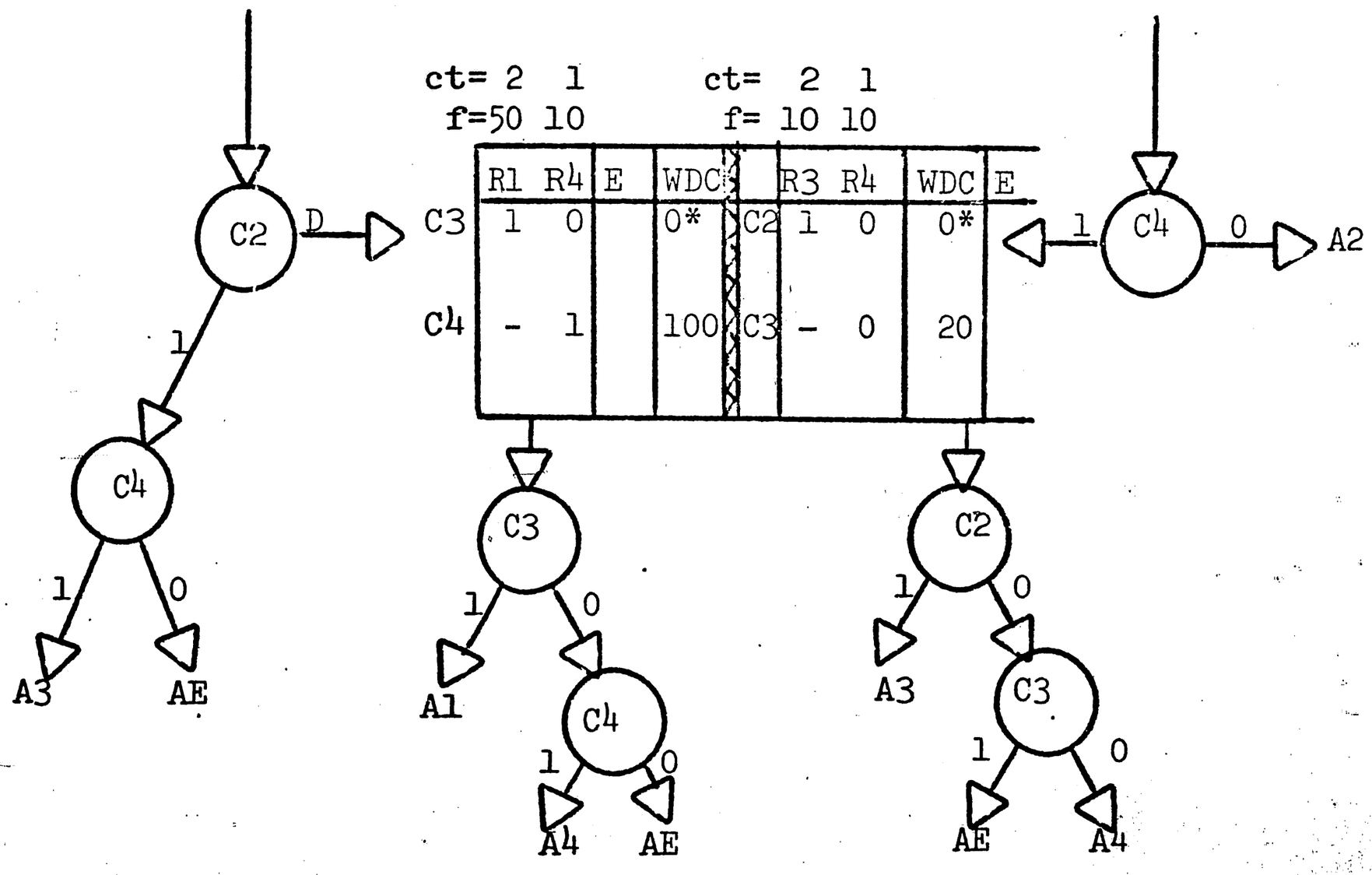


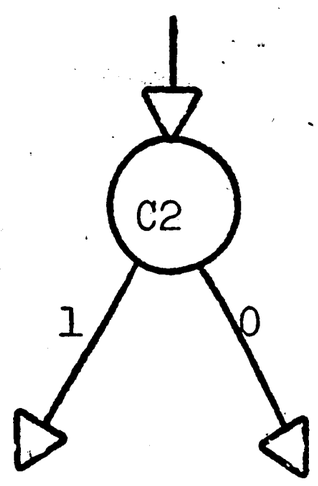
FIGURE 2.7

value of entropy of the non-dash elements of a condition row, where the weight (P) corresponds to the probability that the condition will be pertinent to an arriving transaction (i.e., P is the sum of the probabilities of the rules in the table whose entries for the condition are not dashes).

Shwayder offered no justification for his algorithm, but research by Garey[5] does provide us with at least an intuitive feel for its usefulness as a heuristic approach. Consider a table with no dashes (all simple rules), where each condition cost (time) is one and the procedure of constructing a decision tree is to always choose the condition which splits the remaining rule possibilities (the two resulting sub-tables) into sets having the most nearly equal probability sums. It can be shown that this "probability splitting" algorithm yields a decision tree whose average response time differs, at most, by one from the optimal (minimum) response time. For this particular type of table the entropy (E) calculation will always be maximized when this probability splitting is accomplished (i.e., when $P_1 = .5$ and $P_0 = .5$ $E = 1 = max.$). Therefore, we see that the use of the entropy calculation is nothing more than an approach to "probability splitting" even when there are dashes in a table. But for the general case, where the rules may have dashes, it is not known whether the ratio of the cost of a probability splitting algorithm's solution to the optimal cost, for tables with unit condition testing costs, is bounded by a constant[5]. Application of Shyayder's algorithm to a decision table is illustrated in Figure 2.8.

Rule
Freq. 1/16 4/16 2/16 3/16 5/16 1/16

	R1	R2	R3	R4	R5	R6	P	P_1	P_0	E	
C1	1	0	1	1	0	1	1	7/16	9/16	.990	
C2	1	1	1	0	0	1	1	1/2	1/2	1.00	*Max.
C3	1	-	0	-	-	1	1/4	1/2	1/2	.250	
C4	0	-	-	-	-	1					
	A1	A2	A3	A4	A5	A6					

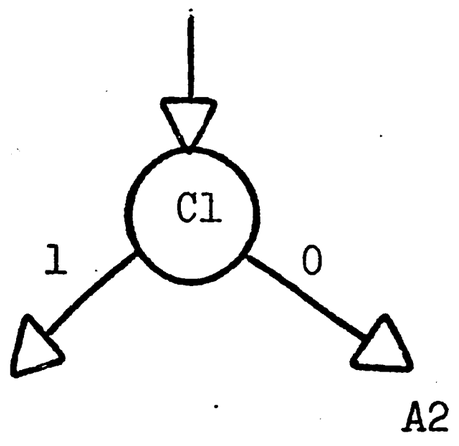


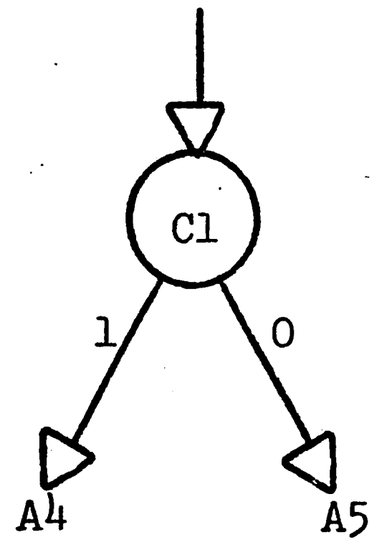
1/8 4/8 2/8 1/8

	R1	R2	R3	R6	P	P_1	P_0	E	
C1	1	0	1	1	1	1/2	1/2	1.0	
C3	1	-	0	1	1/2	1/2	1/2	.50	*
C4	0	-	-	1	1/4	1/2	1/2	.25	
	A1	A2	A3	A6					

3/8 5/8

	R4	R5	P	P_1	P_0	E	
C1	1	0	1	3/8	5/8	.953	*
C3	-	-	0	N/A	N/A	0	
C4	-	-	0	N/A	N/A	0	





1/4 2/4 1/4

	R1	R3	R6	P	P_1	P_0	E	
C3	1	0	1	1	1/2	1/2	1.0	*
C4	0	-	1	1/2	1/2	1/2	.50	

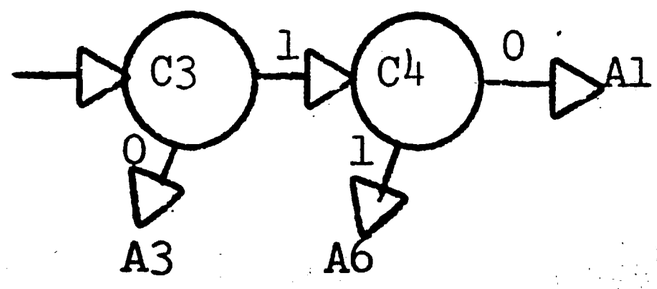


FIGURE 2.8

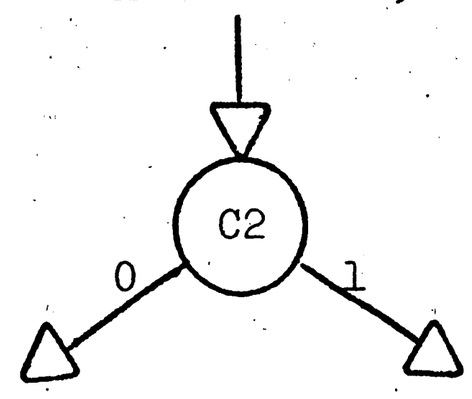
Verhelst[23] also proposed an algorithm for the minimum response time criterion. As in previous algorithms it does not allow for an ELSE rule, and is not accompanied with any theoretical background or justification. The algorithm does allow the user to specify different costs (time) for the testing of conditions. The underlying decision rule is to always choose a condition to test that is most pertinent. In this case the lower the frequency of rules with dash entries for a particular condition, the more pertinent it is. This value of pertinency is then "weighted" by the cost (time) of testing the condition, giving cost-pertinency trade offs in the final calculation. If f_j is the frequency of the j^{th} rule in the table and t_i is the cost (time)associated with testing the i^{th} conditon, then at each stage in the selection of a condition i, choose i such that:

$$T_i = t_i \sum_j f_j \quad \text{is minimized.}$$

(j for all rules containing a dash in condition i) As we will see in Chapter III, a dash does not <u>guarantee</u> that a condition will not have to be tested in order to identify a particular action set to be executed. Therefore, any calculation that bases its results (as in this algorithm and Pollack's) on the dashes, without investigating which dashes will result in the elimination of condition tests, will not be as accurate as one that does. The algorithm by Verhelst is demonstrated in Figure 2.9.

	R1	R2	R3	R4	R5		
f_i	10	30	5	20	35	t_i	$T_i = t_i \sum_j f_j$
C1	0	-	1	0	1	60	180
C2	0	1	-	1	0	5	25 *Min.
C3	-	0	1	1	0	11	110

A1 A2 A3 A4 A5

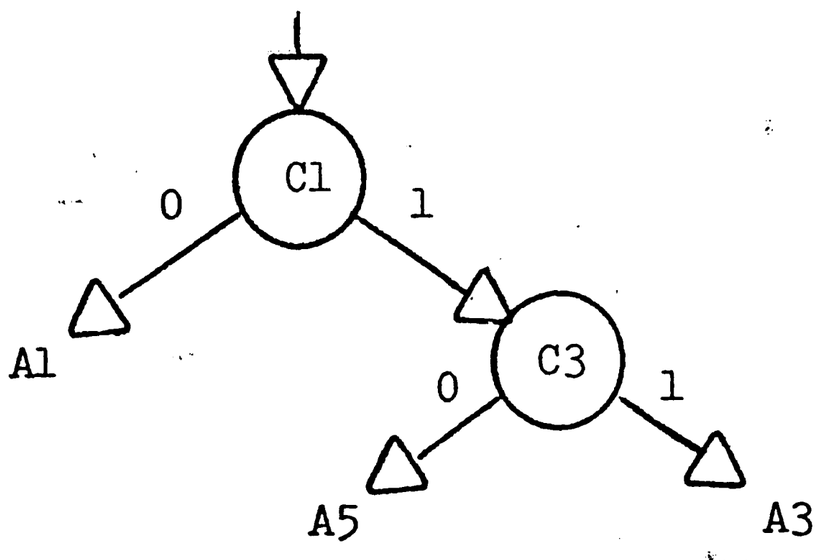


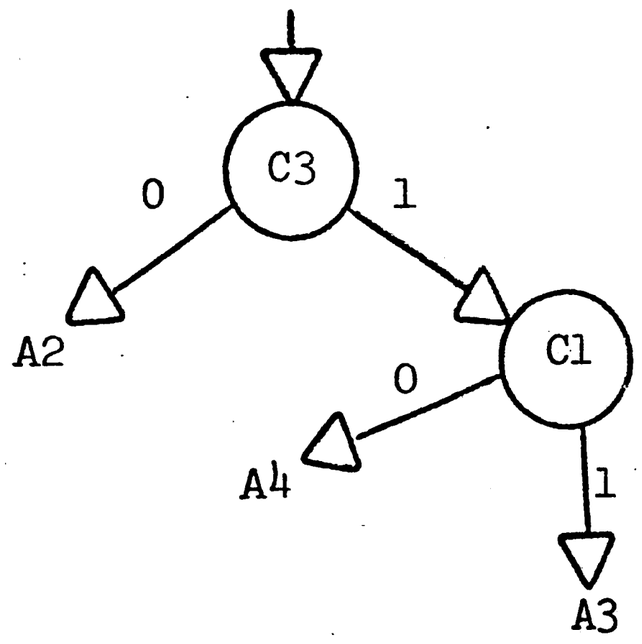
f_i	10	5	35	t_i	T_i	
C1	0	1	1	60	0	*
C3	-	1	0	11	110	

A1 A3 A5

	30	5	20	t_i	T_i	
C1	-	1	0	60	180	
C3	0	1	1	11	0	*

A2 A3 A4





Avg. Response Time = .1(25+180) + .35(25+180+110) + .05(25+180+110)
+ .30(25+110) + .20(25+110+180) + .05(25+110+180)

= .1(205) + .35(315) + .05(315) + .30(135)
+ .20(315) + .05(315) = 265.75

FIGURE 2.9

CHAPTER III

OPTIMIZATION WITH MINIMUM STORAGE CONSIDERATIONS

A. Definitions and Terminology

Before discussing the fundamental concepts of the algorithms, it seems advisable to define the terms that are used.

(1) Binary Decision Tree - A rooted binary tree having the following properties:

a. The non-terminal nodes are conditions,
b. The terminal nodes are pointers to sequences of actions, and
c. No condition appears more than once along a path from the root node to a terminal node.

(2) Complete Decision Tree - A binary decision tree having exactly 2^n terminal nodes, where n is the number of conditions.

(3) Partial Decision Tree - A binary decision tree having less than 2^n terminal nodes, where n is the number of conditions.

(4) Conqruent Nodes - Any set of two or more non-terminal nodes such that each node in the set has exactly the same number of non-terminal nodes between the root of the complete tree and itself.

(5) Level Number in a Decision Tree - The number, k, associated with a set of conqruent nodes where k - 2 is the number of non-terminal nodes between the root node of the complete tree, and any node in the set.

The root node is defined to be at level one.

(6) Sub-Tree - A binary decision tree whose root node is a non-terminal node of some other binary decision tree. A sub-tree may be considered complete or partial.

(7) Sub-Table - A decision table generated from some other decision table or sub-table by deleting a condition row and the corresponding rules for either the conditions zero's or one's group.

(8) Node Set - The name given to the non-terminal nodes in a sub-tree.

(9) Order of a Node Set - The number calculated by the expression $\ln_2 (X)$, where X is the number of terminal nodes in the sub-tree corresponding to the node set.

(10) Parent Node of a Node Set - The root node of the corresponding sub-tree.

(11) Zero's Group of a Condition Row - The term used when referring to both the "zero" and dash elements in a condition row, excluding the "one" elements.

(12) One's Group of a Condition Row - The term used when referring to both the "one" and dash elements in a condition row, excluding the "zero" elements.

(13) A Group of Ones or Zeros - The term used when referring to only the ones or zeros in a condition row, excluding the dash elements.

(14) $\underline{E_{(i)k}}$ - The term used to express the number of non-terminal nodes eliminated due to the ELSE rule, when the i^{th} condition in a table (sub-table) is chosen to test.

(15) $\underline{D_{(i)k}}$ - The term used to express the number of non-terminal nodes eliminated due to the dashes when the i^{th} condition in a table (sub-table) is chosen to test.

(16) Preliminary Maximum - The maximum number of node sets of order i that can be eliminated from a complete tree without regard to row patterns. The elimination of each additional node set of order i requires the existence of unique pairs of rules whose i dashes are in the same row.

B. The Role of the ELSE Rule - Fundamental Concepts

In considering the table in Figure 3.1.1, we see that there are 2^3 simple rules or terminal nodes in a complete tree. The number of possible complete trees is $3^1 \cdot 2^2 = 12$; due to the various ways which the conditions can be tested. This is obtained directly from the general equation for the number of possible complete trees as stated below.

$$\prod_{i=0}^{n-2} (n-i)^{2^i}$$

The size and number of the complete trees are dependent on n, the number of conditions. However, when a table contains dashes and/or an ELSE rule, some of the complete trees can become partial trees. The partial tree with the smallest number of non-terminal nodes is the optimal tree when the objective is to minimize the storage requirements of the object program. Our only assumption will be that the storage required to test each condition is the same.

	R1	R2	R3	R4	R5	R6	R7	R8
C1	0	0	0	0	1	1	1	1
C2	0	1	1	0	0	0	1	1
C3	0	0	1	1	0	1	0	1
	A1	A2	A3	A4	A5	A6	A7	A8

FIGURE 3.1.1

The concept of the complete tree provides the basis for the algorithm. In choosing a condition to be tested at any particular node in the creation of the decision tree; we will choose the condition whose

estimation of the largest number of non-terminal nodes that can be eliminated from the remaining set of complete trees is maximized. Consider the table in Figure 3.1.2. It is obvious that $2^3 - 2$ or 6 simple rules comprise the ELSE rule, Thus, six terminal nodes in any of the complete trees correspond to the ELSE action set (AE). If C1 is chosen as the

	R1	R2	ELSE
C1	0	0	
C2	0	1	
C3	0	1	
	A1	A3	AE

FIGURE 3.1.2

root node, a partial tree as in Figure 3.1.3 will result. We see that $2^2 - 1 = 3$ non-terminal nodes have been eliminated from the set of complete trees that have C1 as their root node. In this case, we have eliminated half of the number of non-terminal nodes in a complete tree. If C1 had been placed elsewhere in a partial tree, the number of non-terminal nodes eliminated due to the fact that the ELSE action set can be executed when C1 = 1, would have been less than half the number in a complete tree.

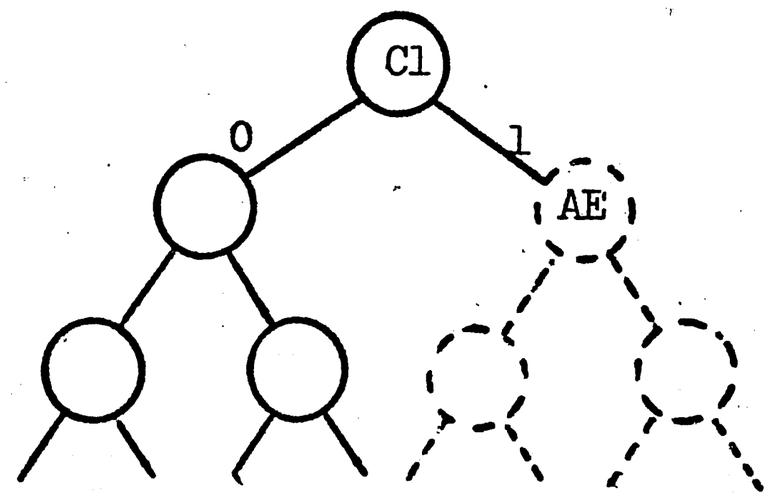


FIGURE 3.1.3

In order to see this, think of the non-terminal nodes that are eliminated when a particular condition is chosen for the root node as a node set, which was defined in the beginning of this Chapter. Since half of the non-terminal nodes are eliminated, obviously half of the terminal nodes ($2^n/2$) are eliminated. By definition, the order of this node set is $\ln_2(2^n/2) = n-1$. Since the number of terminal nodes for the node set is $2^{n-1} = 2^{n-2}\cdot 2 = 2^{n-3}\cdot 2^2 = \ldots = 2^{n-(n-1)}\cdot 2^{n-2}$, we can say that a node set of order n-1 has the same number of terminal nodes as 2^{n-2} node sets of order 1, 2^{n-3} node sets of order 2 and so on. In other words, the elimination of a node set of order n-1 corresponds to the elimination of the same number of terminal nodes as the elimination of 2^{n-2} node sets of order 1, 2^{n-3} node sets of order 2, etc. Since the number of non-terminal nodes eliminated in a node set of order p is $2^p - 1$, for any p, then we see that

$$(2^{n-1}-1) > 2^{n-2}\cdot(2^1-1)$$

$$(2^{n-1}-1) > 2^{n-3}\cdot(2^2-1)$$

$$\vdots$$

$$(2^{n-1}-1) > 2^{n-(n-1)}\cdot(2^{n-2}-1).$$

Thus, if a condition is all ones or all zeros, implying that the opposite state should result in execution of the ELSE action set; testing it first and thereby eliminating half the tree (node set of ordr n-1 or 2^{n-1} non-terminal nodes) will always eliminate the maximum number of non-terminal nodes.

Based on the preceeding discussion, the table in Figure 3.1.4a indicates that C1 should be chosen as the root node. The resulting sub-table is seen in Figure 3.1.4b. Figures 3.1.4c, 3.1.4d and 3.1.4e

	R1	R3	R3	R4	R5	ELSE
C1	1	1	1	1	1	
C2	0	0	1	1	1	
C3	1	1	0	0	1	
C4	0	1	1	0	0	
	A1	A2	A3	A4	A5	AE

FIGURE 3.1.4a

	R1	R2	R3	R4	R5	ELSE
C2	0	0	1	1	1	
C3	1	1	0	0	1	
C4	0	1	1	0	0	
	A1	A2	A3	A4	A5	AE

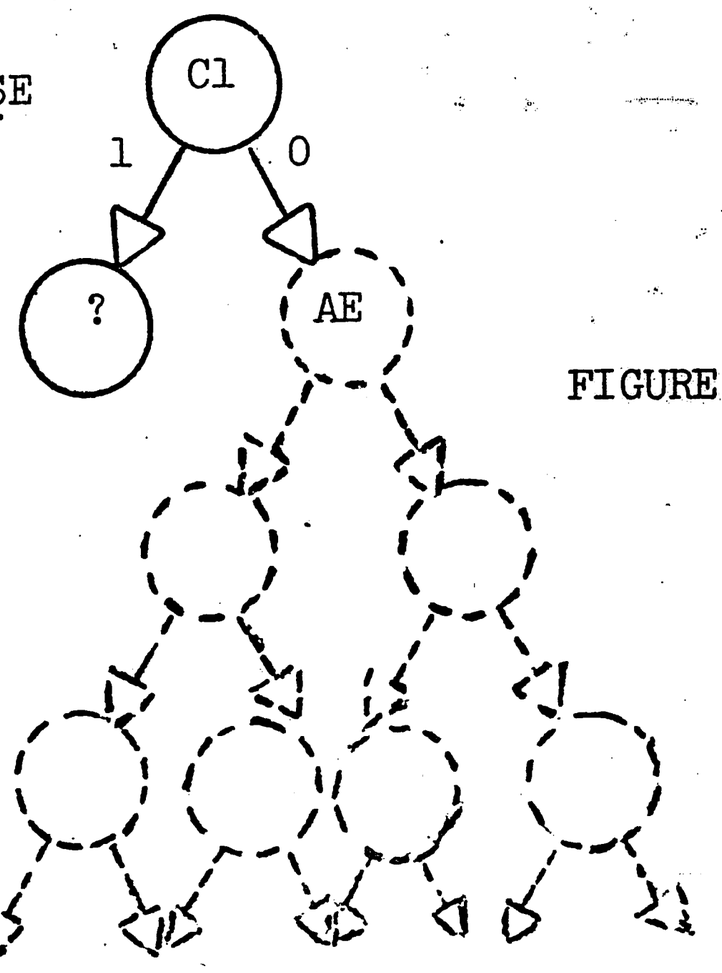
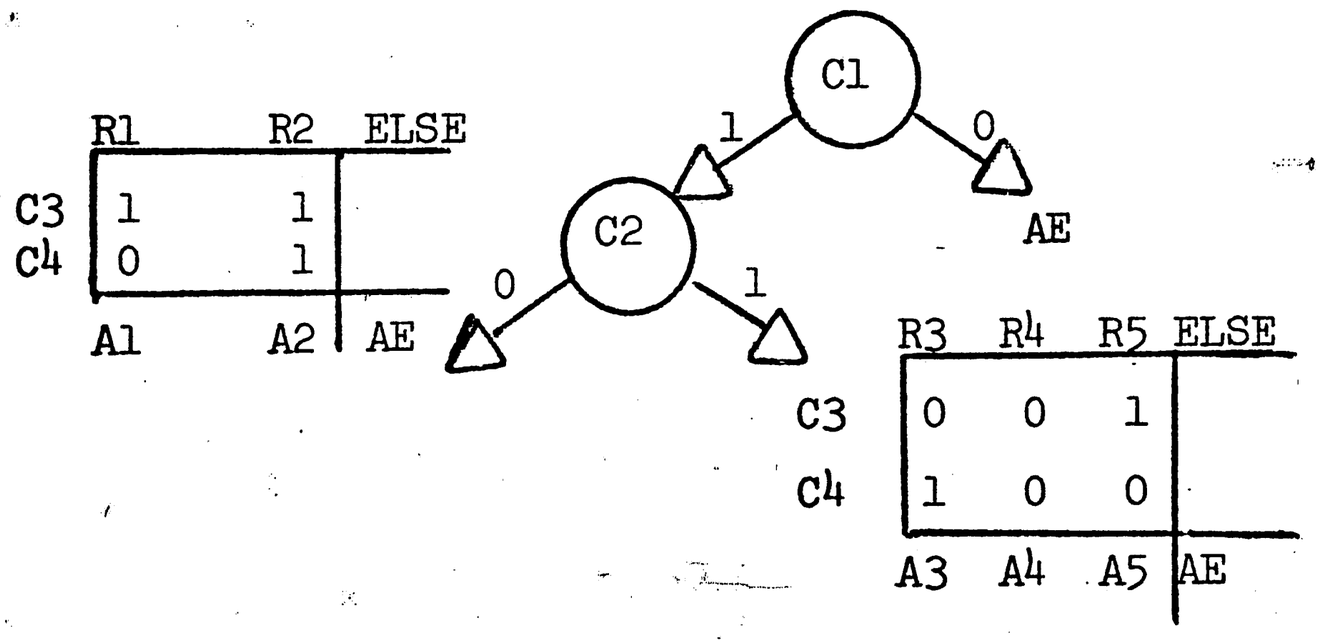


FIGURE 3.1.4b



	R1	R2	ELSE
C3	1	1	
C4	0	1	
	A1	A2	AE

	R3	R4	R5	ELSE
C3	0	0	1	
C4	1	0	0	
	A3	A4	A5	AE

FIGURE 3.1.4c

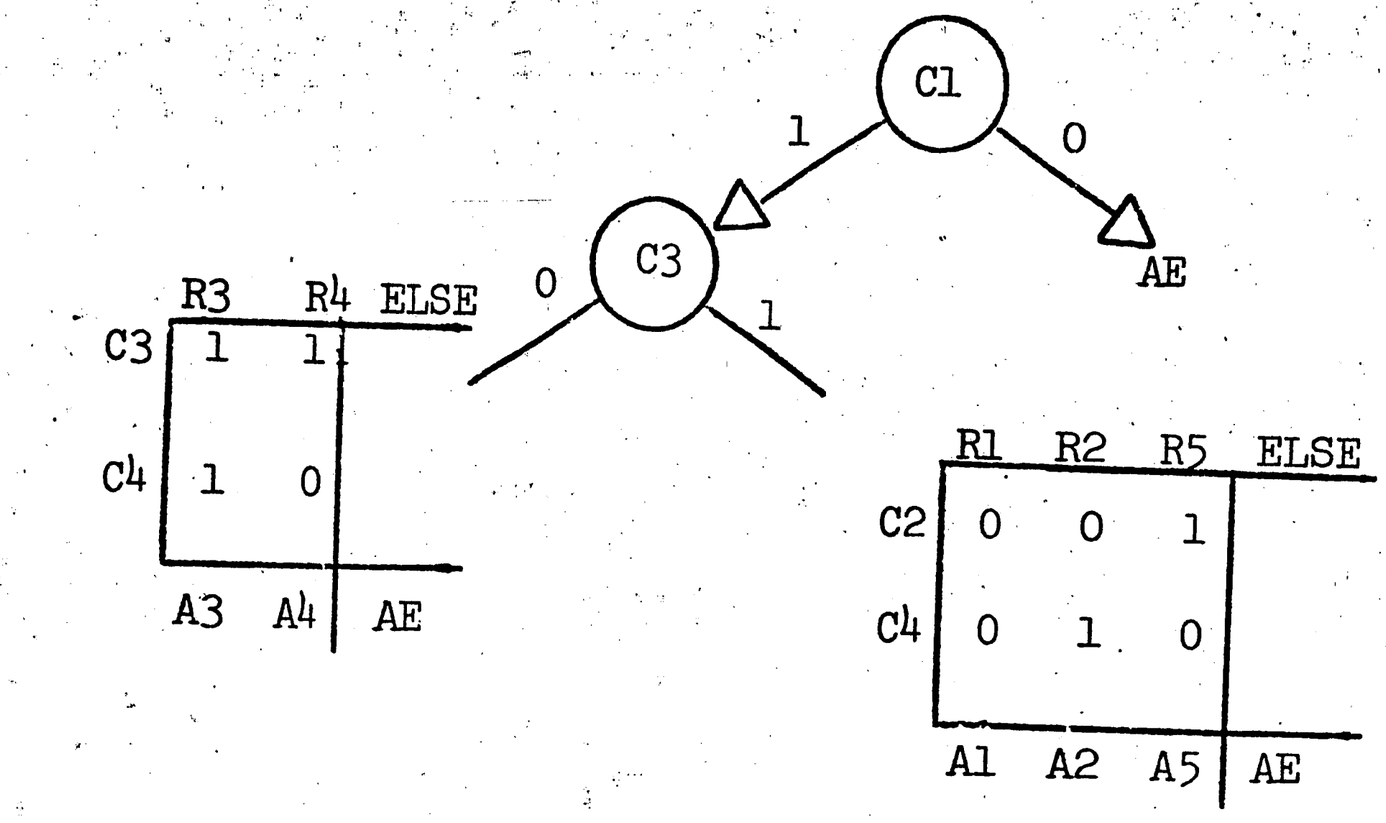


FIGURE 3.1.4d

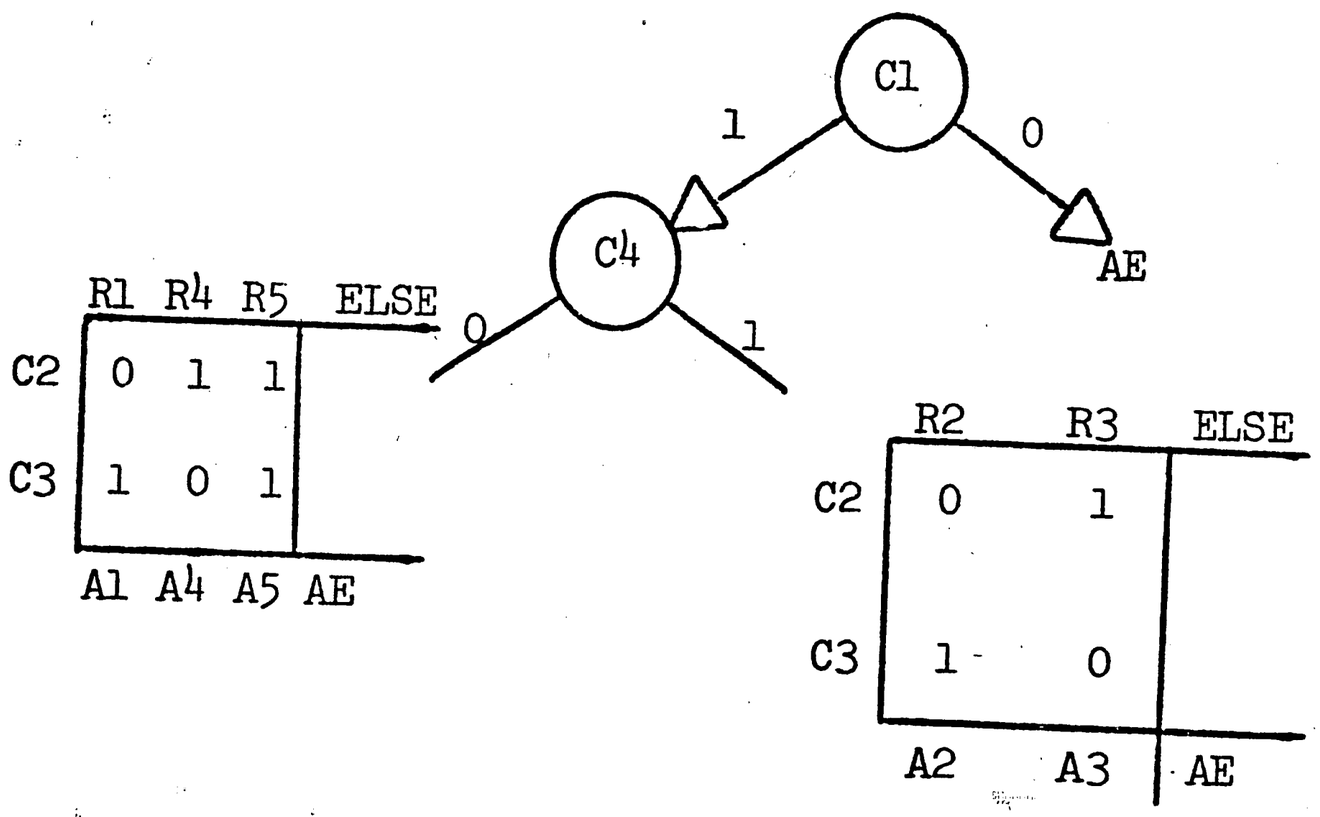


FIGURE 3.1.4e

show the two resulting sub-tables if C3, C2 and C4 are chosen next. By choosing either C2 or C3 off the one's branch of C1, there exists resulting sub-tables that have a condition row of all ones. Namely, C3 for the sub-table off the zero's branch of C2 in Figure 3.1.4c, and C2 for the sub-table off the zero's branch of C3 in Figure 3.1.4d. How do we know which condition to choose in advance? By looking at the sub-table off the one's branch of C1 in Figure 3.1.4b, we see that the zero's group of C2 corresponds with a group of ones in C3. Likewise, the zeros group of C3 corresponds with a group of ones in C2. Condition C4 has no such relationship. This "look-ahead" strategy provides for the detection of future sub-tables whose condition rows have all ones or all zeros. Based on our "look-ahead" strategy, conditions C2 and C3 appear to be better choices than C4 for the node off the one's branch of C1 in the creation of the tree.

We now note that the "look-ahead" strategy could provide for the detection of a correspondence between more than <u>two</u> condition rows. To illustrate this, consider the table in Figure 3.1.5. The one's group of C1 corresponds with all zeros in rows C2 and C3. Likewise, the zero's group of C1 corresponds with all ones in row C4. The total number of these row correspondences for C1 is three; two for its one's group and one for its zero's group. It also can be seen that C2 has a total of two row correspondences; C3, a total of two; and C4, a total of three. Since both C1 and C4 have a total of three row correspondences, it might appear that they are equal candidates for the root node of the decision tree. This, however, is not the case.

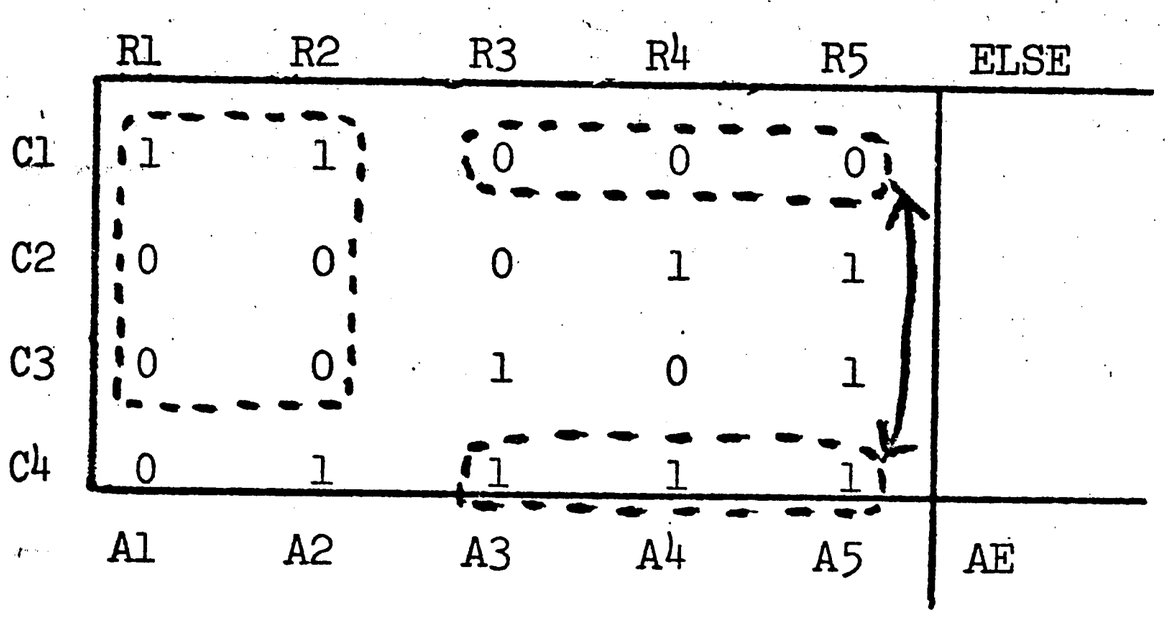

R1
R2
R3
R4
R5
ELSE
C1
C2
C3
C4
1 1 0 0 0
0 0 0 1 1
0 0 1 0 1
0 1 1 1 1
A1
A2
A3
A4
A5
AE

FIGURE 3.1.5

In Figure 3.1.6 a decision tree with C1 as the root node is illustrated for the table in Figure 3.1.5. The terminal node AE01 denotes the ELSE action set when C1 and C4 are both zero, and this can be foreseen by the correspondence of C1's zeros group to a group of ones in C4. The terminal node AE11, also denoting the ELSE action set, can be foreseen by the correspondence of C1's ones group to a group of zeros in C4. Similarly, AE12 is foreseen by the correspondence of C1's ones group to a group of zeros in C3. Thus, we can say:

(1) AE11 is associated with the <u>first</u> correspondence for C1's ones group that results in the elimination of non-terminal nodes from a complete tree.

(2) AE12 is associated with the <u>second</u> correspondence for C1's ones group that results in an elimination.

(3) AE01 is associated with the <u>first</u> and only correspondence for C1's zeros group that results in an elimination.

It can be seen in Figure 3.1.6 that AE11 and AE01 replaced three non-terminal nodes in a complete tree. AE12 replaced only one non-terminal node. When C1 is chosen as the root node, the total number of non-terminal nodes eliminated from a complete tree due to the ELSE action set (AE) and forseen by the row correspondence in the table, is $3 + 3 + 1 = 7$.

Condition C4 <u>also</u> had three row correspondences; one <u>first</u>, one <u>second</u> and one <u>third</u>. Each with its zero's group. The associated number of non-terminal nodes eliminated from a complete tree, when C4

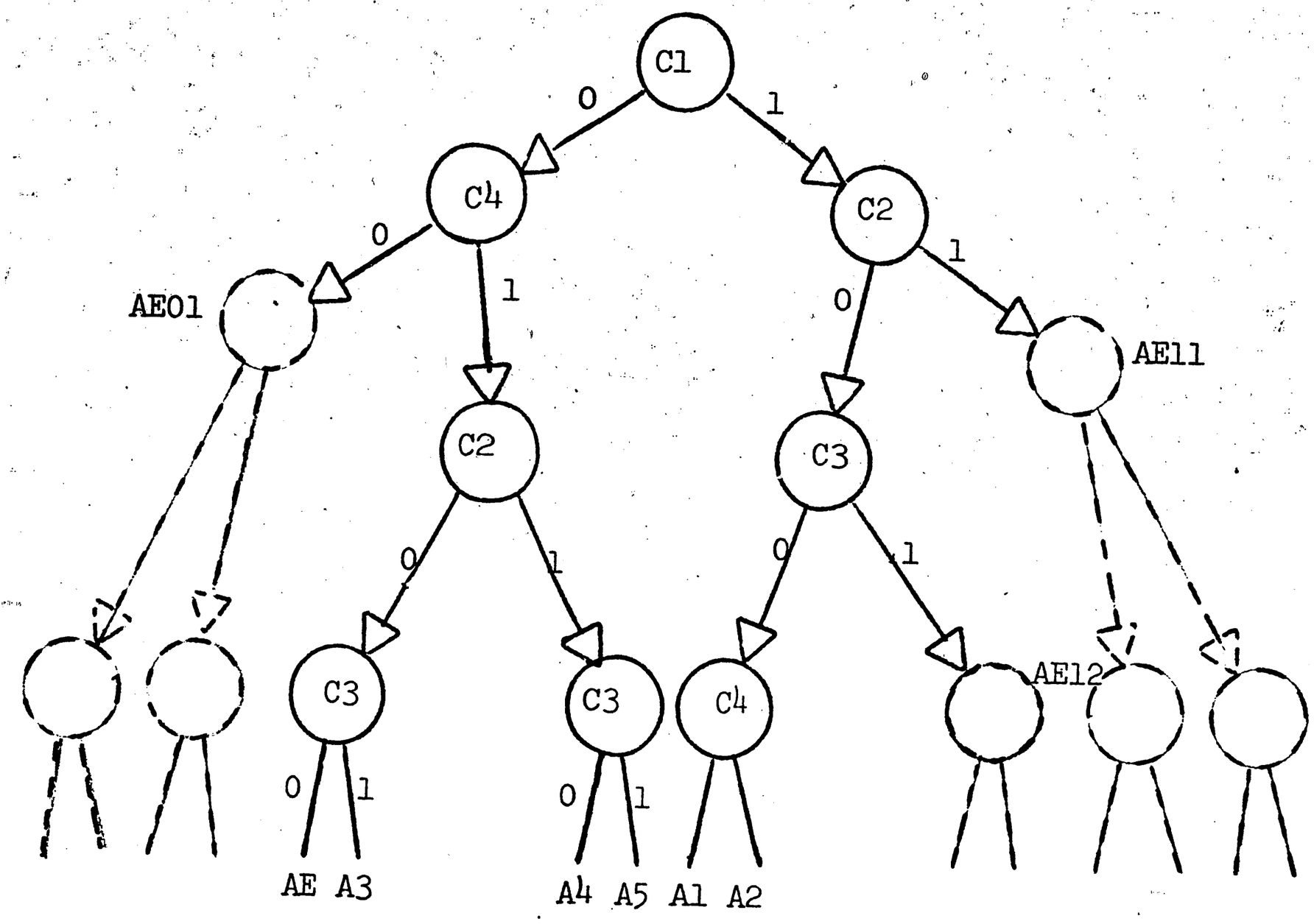
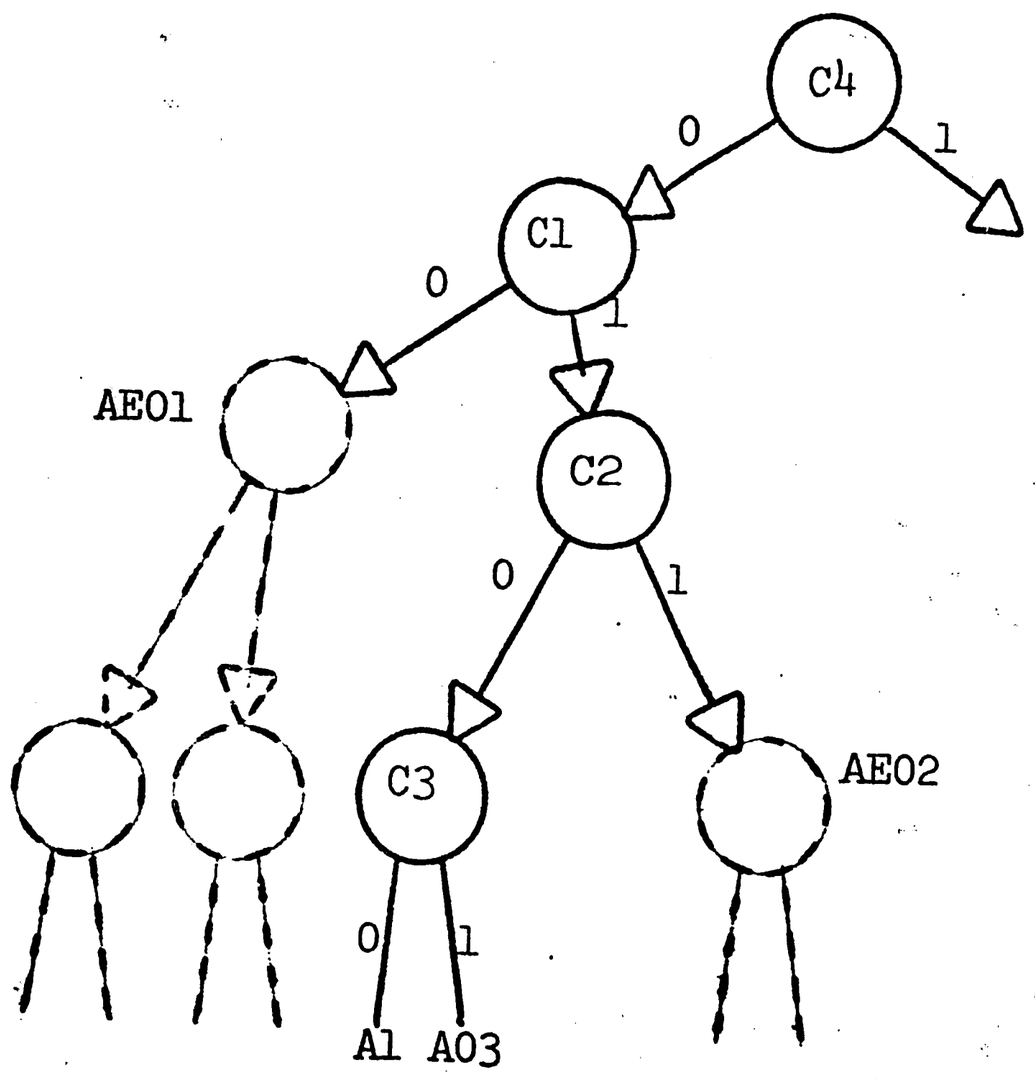


FIGURE 3.1.6



	R2	R3	R4	R5	ELSE
C1	1	0	0	0	
C2	0	0	1	1	
C3	0	1	0	1	
	A1	A2	A3	A4	AE

FIGURE 3.1.7

is the root node, is 3 + 1 = 4, as seen in Figure 3.1.7. The third row correspondence did not result in an elimination since AE03 was itself a terminal node. C1 is obviously the better choice for the root node of the decision tree; and we see that it does not suffice to just count the total number of row correspondences that a condition has, but to distinguish between first correspondence, second, third, etc. The number of non-terminal nodes eliminated due to the ELSE action set and foreseen by the row correspondences in the table can be expressed as follows:

$$E = \sum_{r=0}^{l} \left[\sum_{j=1}^{CR_r} (2^{m-j-1}) - CR_r \right]$$

where CR_r = the number of row correspondences for the r's group of the condition in consideration.

m = the number of conditions in the table.

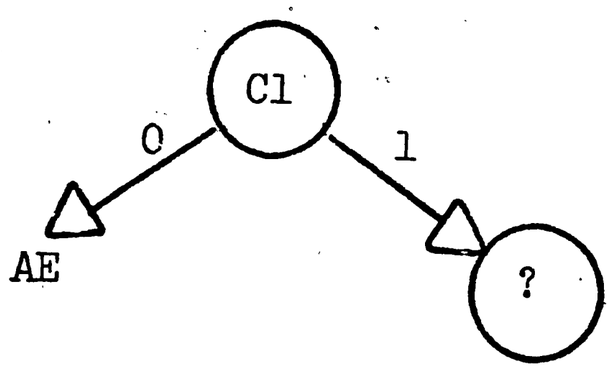
As an example, E for C1 is Figure 3.1.5 is:

$$\left[(2^{4-1-1} + 2^{4-2-1}) - 2\right] + (2^{4-1-1} - 1) = 7.$$

This is verified in Figure 3.1.6. Since E can be recursively calculated for all the conditions in the original table and any sub-table at some level k in the development of a decision tree; the general expression becomes:

$$E_{(i)k} = \sum_{r=0}^{l} \left[\sum_{j=1}^{CR_{(i)r}} (2^{m-j-1}) - CR_{(i)r} \right]$$

(Note that (i) denotes the i^{th} condition in a table or sub-table, and that $\sum_{1}^{0} = 0$). Consider the example in Figure 3.1.8, where the root node



	R1	R2	R3	R4	R5	ELSE
*C2	1	1	0	0	0	
C3	0	0	0	1	1	
C4	0	0	1	0	1	LEVEL 2
C5	0	1	1	1	1	
	A1	A2	A3	A4	A5	AE

*max $E_{(i)k}$

FIGURE 3.1.8

has been determined (C1). In choosing the condition to be tested off the one's branch of C1 at level 2, the following calculations are made:

$$E_{(1)2} = (2^{5-1-2} - 1) + \left[(2^{5-1-2} + 2^{5-2-2}) - 2\right] = 7$$

$$E_{(2)2} = (2^{5-1-2} + 2^{5-2-2}) - 2 = 4$$

$$E_{(3)2} = (2^{5-1-2} + 2^{5-2-2}) - 2 = 4$$

$$E_{(4)2} = (2^{5-1-2} + 2^{5-3-2} + 2^{5-3-2}) - 3 = 4$$

The number seven for $E_{(1)2}$ is a lower bound for the number of non-terminal nodes that can be eliminated from a complete tree having C1 as its root node. The $E_{(i)k}$ calculations are lower bounds due to the fact that the "look ahead" strategy, by means of the row correspondences, only looks for the largest or half tree eliminations. Other smaller eliminations are not investigated due to the many combinations of testing sequences that can exist; thus, the heuristic nature of the calculations. An entire tree is obtained from the recursive application of the $E_{(i)k}$ calculations to all sub-tables; choosing the condition to test from each sub-table whose $E_{(i)k}$ value is the largest. The tree appears in Figure 3.1.9

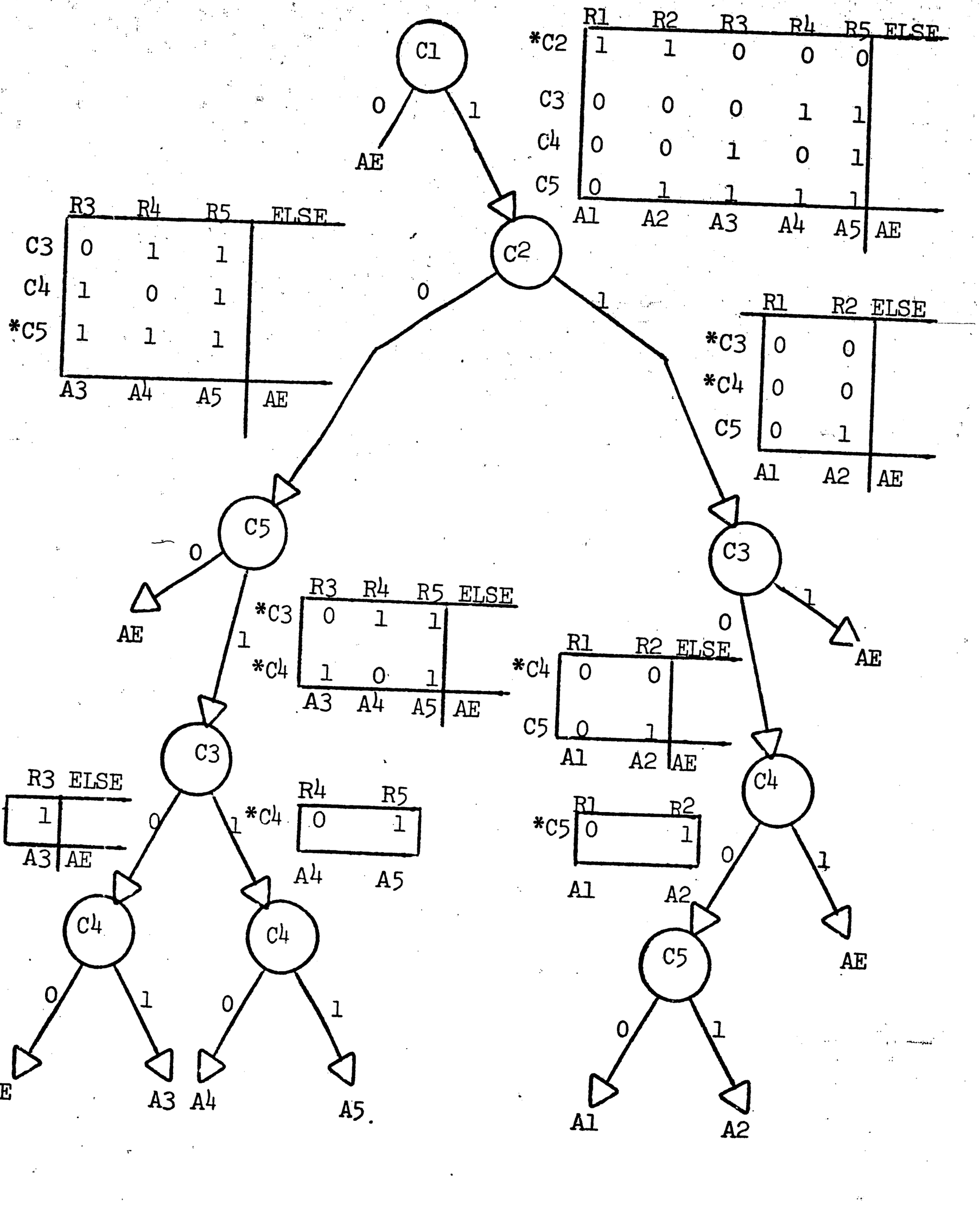

C1
0
1
AE
C2
0
1
R1 R2 R3 R4 R5 ELSE
*C2 1 1 0 0 0
C3 0 0 0 1 1
C4 0 0 1 0 1
C5 0 1 1 1 1
A1 A2 A3 A4 A5 AE
R3 R4 R5 ELSE
C3 0 1 1
C4 1 0 1
*C5 1 1 1
A3 A4 A5 AE
R1 R2 ELSE
*C3 0 0
*C4 0 0
C5 0 1
A1 A2 AE
C5
0
1
AE
R3 R4 R5 ELSE
*C3 0 1 1
*C4 1 0 1
A3 A4 A5 AE
C3
0
1
AE
R1 R2 ELSE
*C4 0 0
C5 0 1
A1 A2 AE
C3
R3 ELSE
*C4 1
A3 AE
R4 R5
*C4 0 1
A4 A5
C4
C4
C4
0
1
0
1
0
1
AE
A3
A4
A5
R1 R2
*C5 0 1
A1
C4
A2
C5
AE
0
1
A1
A2

FIGURE 3.1.9

C. The Role of the Dash-Fundamental Concepts

1. Single Dash Per Rule - No ELSE Rule

Since the occurrence of dashes in a decision table is common; not only does the $E_{(i)k}$ have to be calculated but also some $D_{(i)k}$ which considers the role of the dashes in node elimination. Once $D_{(i)k}$ has been calculated we would choose the condition Ci to test which gave the $\max(E_{(i)k} + D_{(i)k})$. In formulating the expression for $D_{(i)k}$, first consider the table:

	R1	R2	R3	R4	R5	R6
C1	-	0	1	1	0	1
C2	0	-	1	1	1	0
C3	0	1	1	0	0	1
	A1	A2	A3	A4	A5	A6

FIGURE 3.2.1

A rule with a dash is the combination of two simple rules. A condition corresponding to a dash is not pertinent to identifying the rule with the dash and may not have to be tested. In Figure 3.2.2, if C1 is chosen as the root, then all three conditions will have to be tested to identify R1 from the table in Figure 3.2.1. We have, in a sense, used up any advantage we might have had in minimizing storage due to the dash in C1. But we see, in Figure 3.2.2, that if C3 is chosen off the zero's branch of C1, then C2 does not have to be tested to identify R2. This has the effect of eliminating $(2^1-1) = 1$ non-terminal node from the complete tree. The same number of non-terminal nodes, (2^1-1), would also have been eliminated if C2 were chosen as the root node followed by C3 on its zero's branch.

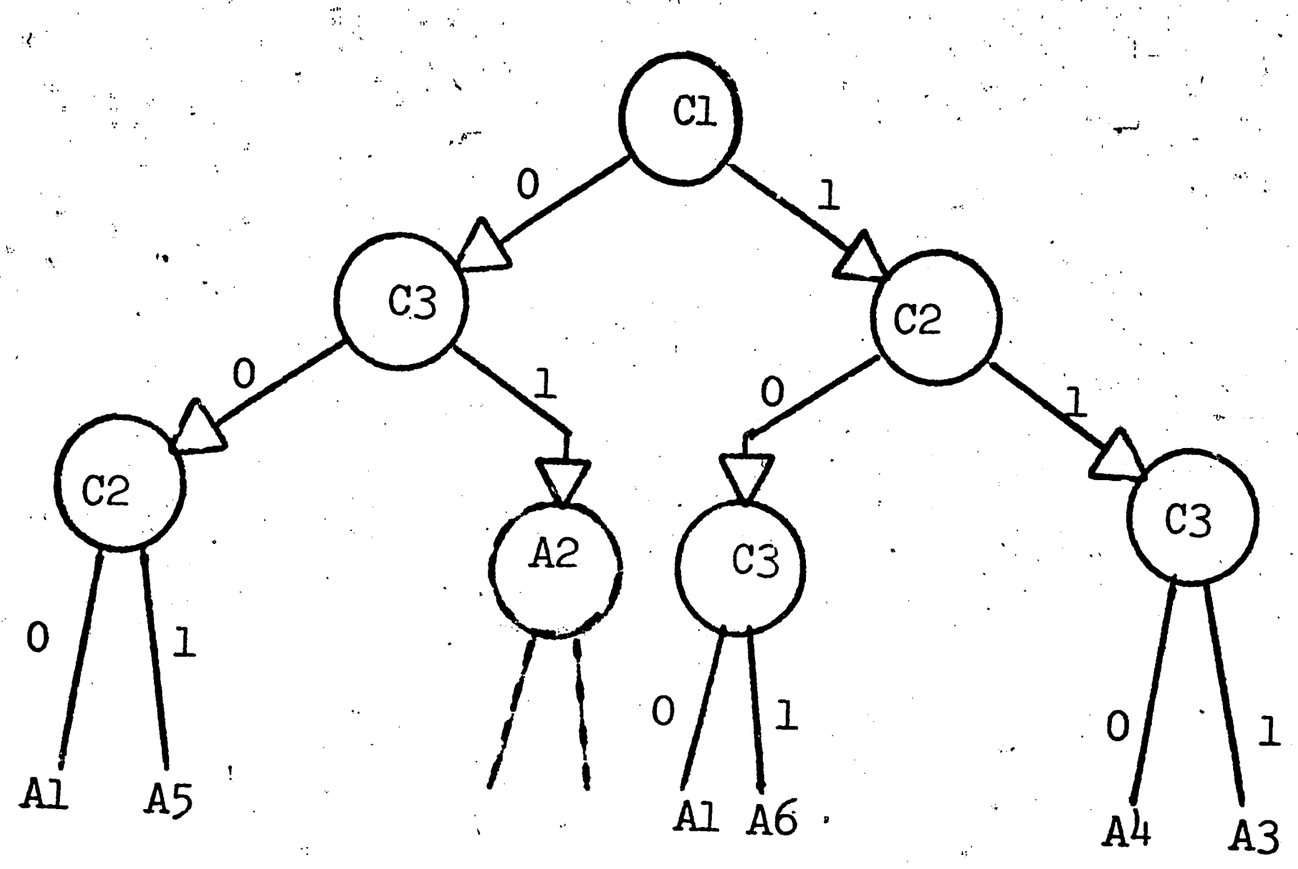


FIGURE 3.2.2

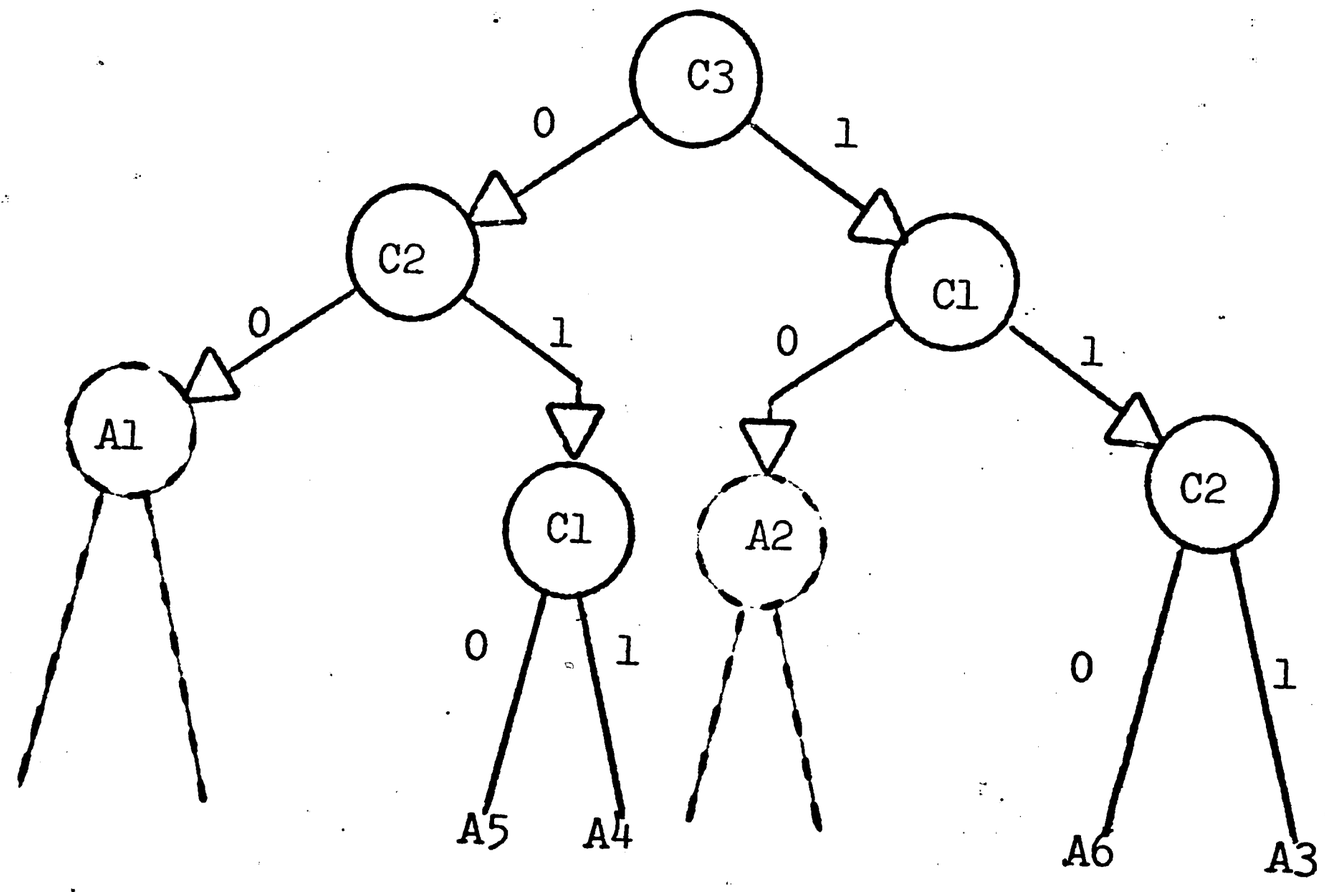


FIGURE 3.2.3

However, looking at Figure 3.2.3, the choosing of C3 for the root node results in the elimination of $(2^1-1) + (2^1-1) = 2$ non-terminal nodes. Thus, C3 would have been a better choice for the root of the decision tree. It, in fact, yields the smallest tree.

So far, only one example table has been considered and there is more to consider in the investigation of the dashes. Consider the table in Figure 3.2.4:

	R1	R2	R3	R4	.....
C1	0	1	-	0	
C2	0	-	1	0	Rest of Simple rules
C3	-	0	1	0	
C4	1	1	1	0	
	A1	A2	A3	A4	.....

FIGURE 3.2.4

Based on the previous observations, C4 would appear to offer an elimination of $(2^1-1) + (2^1-1) + (2^1-1) = 3$ non-terminal nodes if chosen as the root node. As seen in Figure 3.2.5, there are four non-terminal nodes at the lowest level in the tree that are candidates for elimination off the one's branch of C4. This is because a single dash in a rule implies the combination of two simple rules, thus allowing for the possible elimination of $(2^1-1) = 1$ non-terminal node. If three of the four non-terminal nodes (designated by the X's in Figure 3.2.5) are eliminated, then two of the three must have the same parent node. This forces a particular pattern on the dashes in the table in Figure 3.2.4, which can be described as follows: The two rules in the table whose

Node sets of order 1

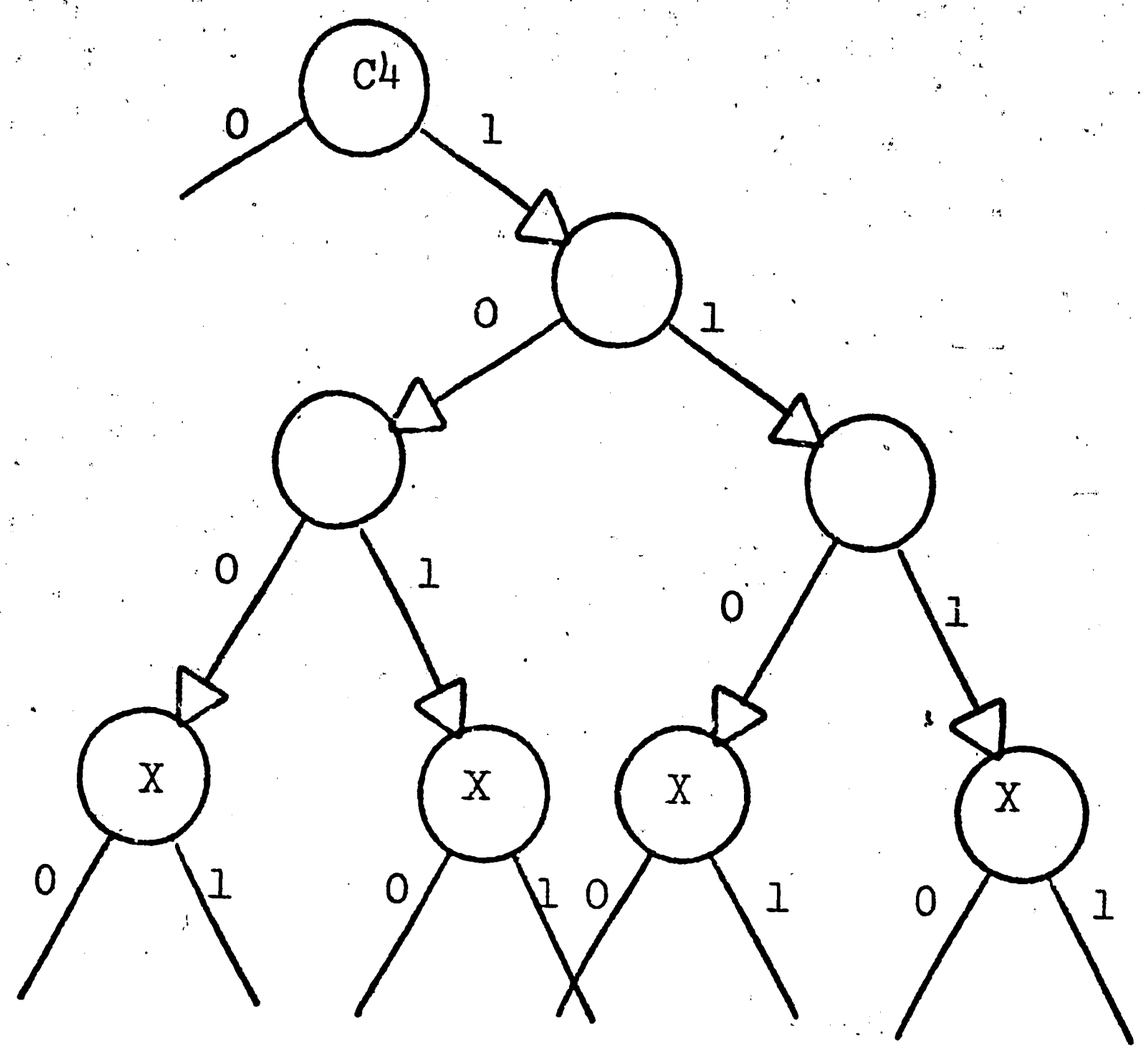


FIGURE 3.2.5

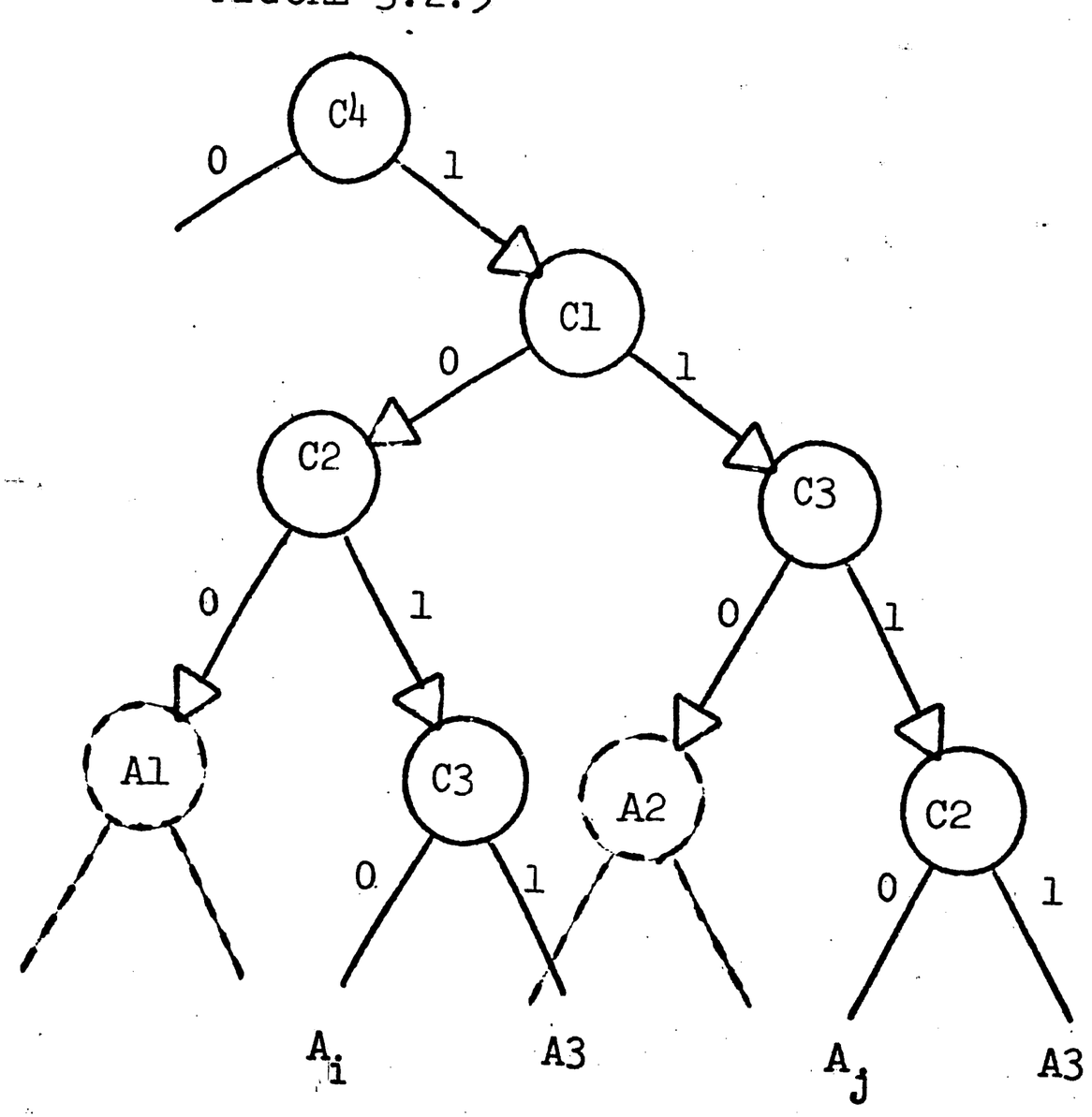


FIGURE 3.2.6

corresponding paths in the tree pass through this parent node must be the same with one exception. This is that they differ in only one condition, namely the one existing at the parent node. Also,

> the two dashes that correspond to the elimination of the two non-terminal nodes with this parent node must be in the same condition row in the decision table; otherwise, dash splitting will occur.

We can now define the preliminary maximum (PM) number of node sets of order one that can be eliminated from the complete tree, off either the zero's or one's branch of the root node, to be $2^{m-2}/2$. This number applies to tables with single dashes in the rules. In addition, it is the maximum number of node sets of order one that can be eliminated without the structure of the tree forcing the previously mentioned pattern on the table.

The example table in Figure 3.2.4 has three dashes associated with the one's group of C4, but none of the dashes are in the same row. Therefore, the maximum number of non-terminal nodes (node sets of order one) that can be eliminated is $2^2/2 = 2$, which is the value of the preliminary maximum. This tells us that one of the three dashes will not result in an elimination of a non-terminal node. The condition corresponding to the row the dash is in will have to be tested, even though it is irrelevant to the rule in which the dash appears. We say that the dash has been "split". This is seen in Figure 3.2.6, where C1 must be tested in determining A3, even though C1 is irrelevant to R3.

Let us choose to call a condition's zeros or ones group, its r's group (r = 0,1). Then, designating the number of rules corresponding to the condition's r's group that have single dashes as P_1, we may state the following for tables <u>with single dashes</u> per rule and no ELSE rule:

(1) For any given non-terminal node at level k in the creation of a decision tree; if for a condition being considered for this node, $P_1 \leq 2^{m-1-k}/2$ (PM) then $P_1 \bullet (2^1-1)$ is the number of non-terminal nodes that can be eliminated from the complete tree off the r's branch of the condition. In this case there is no dash splitting.

(2) If $P_1 > 2^{m-1-k}/2$ (PM) and (a) none of the P_1 rules have their dashes in the same row, or

(b) given that rules do exist with their dashes in the same row, these rules differ in more than one condition row; then $(2^{m-1-k}/2) \bullet (2^1-1)$ is the number of non-terminal nodes that can be eliminated from the complete tree. In this case $P_1 - (2^{m-1-k}/2)$ dashes are split.

(3) If $P_1 > (2^{m-1-k}/2)$ and unique pairs of the P_1 columns (rules) whose dashes are in the same row and who differ in only one condition do exist, then for each such pair the PM value $(2^{m-1-k}/2)$ may be increased by one. If C is the number of those pairs, then $(2^{m-1-k}/2)+C \bullet (2^1-1)$ is the number of non-terminal nodes that can be eliminated from the complete tree. In this case only $P_1 - (2^{m-1-k}/2+C)$

dashes are split. Consider the table in Figure 3.2.7 and assume k = 1.

	R1	R2	R3	R4	R5	R6	.....
C1	-	0	0	1	1	0	
C2	0	-	0	1	0	0	
C3	0	1	-	0	1	1	Rest of Simple
C4	0	0	1	-	-	1	Rules
C5	1	1	1	1	1	0	
	A1	A2	A3	A4	A5	A6	.....

FIGURE 3.2.7

For C5 one's group, $P_1 = 5$. That is, there are five rules that have one dash in rows other than C5. Since m = 5 and k = 1, then $2^{m-1-k}/2 = 4$, and 4 becomes the preliminary maximum for the number of node sets of order one that can be eliminated from the complete tree if C4 is chosen as the root node. Obviously, $P_1 = 5 > 4$ and there does exist a unique pair of rules (columns), namely R4 and R5, whose dashes are in the same condition row (C4). However, R4 and R5 differ in more than one condition row. Therefore, we calculate $4(2^1-1) = 4$ non-terminal nodes that can be eliminated. For C5's zero's group $P_1 = 0$, which implies that no non-terminal nodes can be eliminated. An interesting point here is that C1, C2 and C3 all give the same calculated number of 4 non-terminal nodes to be eliminated. This means that they are equally good choices for the root node along with C5. However, existing algorithms consistently choose C5 as being the only choice. Thus, we see that the foundations of a more thorough analysis are being laid without any significant increase in translation time for the creation of the decision tree. The main idea of the analysis is the recognition of the fact that a decision table may necessitate the splitting of dashes.

The preceeding three observations applied to tables with: (1) a single dash per rule and (2) no ELSE rule. They do, however, provide the foundation for the development of the $D_{(i)k}$ in the general case where tables may have one or more dashes in the rules and may have an ELSE rule. It is the general case where we combine the $D_{(i)k}$ and $E_{(i)k}$ in order to make a decision on which condition to choose at each step in the creation of the decision tree.

2. Double Dashes Per Rule - No ELSE Rule

Consider a table where the rules may have two dashes, and for the moment assume that there is no ELSE rule. Two dashes in a rule implies that four simple rules have been combined. That is, three non-terminal nodes can be eliminated from the complete tree for a rule with two dashes, if all the other conditions are tested first. The three non-terminal nodes make up a node set of order 2. Thus, we can see the relationship between the number of dashes in a rule and the order of the node set they eliminate. Just as in the single dash case there is a preliminary maximum number of node sets of order 2 that can be eliminated off the r's branch of a condition. This is $(2^{m-2-k}/2)$. For $m = 5$ and $k = 1$, this becomes $2^2/2 = 2$. If more than two are eliminated, say three, as in Figure 3.2.8a, then two of the three must have the same parent node. This implies, as in the single dash case, that two of the three rules having two dashes each, must have the dashes in the same rows in the table and must differ in only one condition.

Considering the table in Figure 3.2.8b we see that $P_2 = 5$ for the zero's group of C6. If it is assumed that $k = 1$ (the condition for the

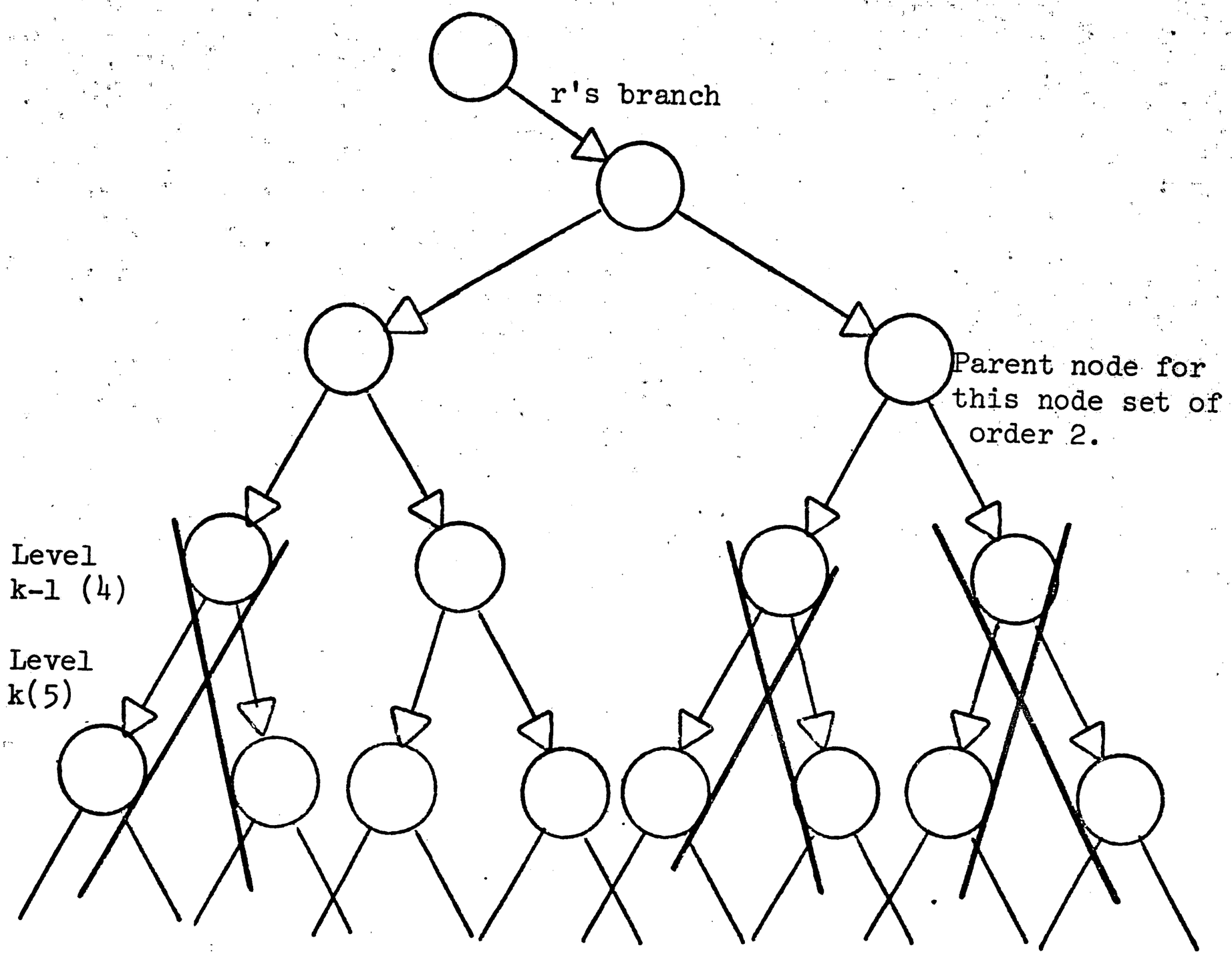


FIGURE 3.2.8a

	R1	R2	R3	R4	R5	R6	ELSE
C1	0	1	1	-	-	0	
C2	0	0	-	-	1	0	
C3	1	-	-	0	1	0	
C4	-	-	0	1	1	0	
C5	-	0	1	1	-	0	
C6	0	0	0	0	0	1	
	A1	A2	A3	A4	A5	A6	AE

FIGURE 3.2.8b

root node is being determined) and since m = 6, we calculate $2^{m-2-k}/2 = 2^3/2 = 4$. This is the prelimanary miximum number of node sets of order two that can be eliminated. Since P = 5 > 4, we look for unique pairs of the rules whose dashes are in the same row. Rules 1 and 2 have this property. However, they do not differ in only one condition row but in two, thus ruling out the possibility of increasing the preliminary maximum of 4 (i.e., c = 0). Since each node set of order two has three non-terminal nodes associated with it (2^2-1), then the calculation for the total number of non-terminal nodes that can be eliminated, if C6 is chosen as the root node, is $D_{(6)1} = 4(2^2-1) = 12$.

3. The General Preliminary Maximum Calculation

The consideration of the $E_{(i)k}$ and $D_{(i)k}$ together in choosing the condition that has the max $(E_{(I)k} + D_{(i)k})$ requires an alteration to the calculation of the preliminary maximum number of node sets of order i that can be eliminated. When calculating the preliminary maximum for a condition whose $E_{(i)k}$ is non-zero (due to the row correspondences parts of the complete tree will be eliminated due to the ELSE rule), we must consider only the portion of the complete tree that will remain after the ELSE rule eliminations have been made. Also, if the table has rules whose number of dashes varies from rule to rule, then we must consider the effect an elimination of a higher order node set has on the preliminary maximum number of lower order node sets for the zero's or one's group of a particular condition.

As we have seen, the first row correspondence for a zero's

or one's group of a condition in the calculation of $E_{(i)k}$, eliminates half of the sub-tree associated with the particular branch in consideration (zero's branch or one's branch). The second row correspondence eliminates half of one of the two smaller sub-trees that comprised the first sub-tree, or the equivalent of one fourth of the first sub-tree. In general, for either the zero's or one's group (branch in the tree) of a condition that is a candidate for the root node of a decision tree: the first one-to-one correspondence eliminates half of the node sets of order m-k-1, m-k-2, ..., 1 off the r's branch of the root node; the second one-to-one correspondence eliminates one fourth the node sets of order m-k-2, m-k-3, ..., 1, and so on. Since $CR_{(i)r}$ was defined to be the number of row correspondences for the r's group of condition i at a particular level in the creation of the decision tree, then $CR_{(i)r}$

$$\sum_{j=1} 1/2^j = 1/2 + 1/4 + \ldots.$$ is the

fraction of node sets of order $m-CR_{(i)r} - k$, $m-CR_{(i)r} - k-1$, ..., 1 that can be eliminated due to the ELSE rule. For tables with <u>single dashes per rule</u> 2^{m-k-1} is the total number of node sets of order one off the r's branch of any node at level k. If k = 1 then the root node is implied. Therefore,

$$\left[\left(1 - \sum_{j=1}^{CR_{(i)r}} 1/2^j\right) \cdot \left(2^{m-k-1}\right)\right] \div 2$$

is the preliminary maximum number of node sets of order one that can be eliminated. Remember that a single dash in a rule corresponds to a node

set of order one. Now we will generalize for a table that has, at most, i dashes per rule. Let j(h) be the order of the highest order node set(s) that can be eliminated off the r's branch of a node when a particular condition is chosen for the node. Here h is the number of distinct orders that exist. This corresponds in a table to the distinct number of dash counts that exist in rules associated with the r's group of some condition in consideration. The preliminary maximum equation for node sets of order j(h) is:

$$MX_{kj(h)} = \max \left\{ \left[M_{kj(h)} \bullet 2^{m-k-j(h)} \right] \div 2,1 \right\}$$

where

$$M_{kj(h)} = 1 - \sum_{j=1}^{CR_{(i)r}} 1/2^{j}$$

$M_{kj(h)}$ is the fraction of node sets of order j(h) left after the ELSE rule eliminations for the r's branch of a node at level k, if the particular condition in consideration is chosen for the node. Now, let j(h-1) be the order of the node sets that can be eliminated next, after the highest order node sets. Our preliminary maximum equation must now take in consideration the effect of node set eliminations of order j(h) on the node sets of order j(h-1) in the tree. The equation becomes:

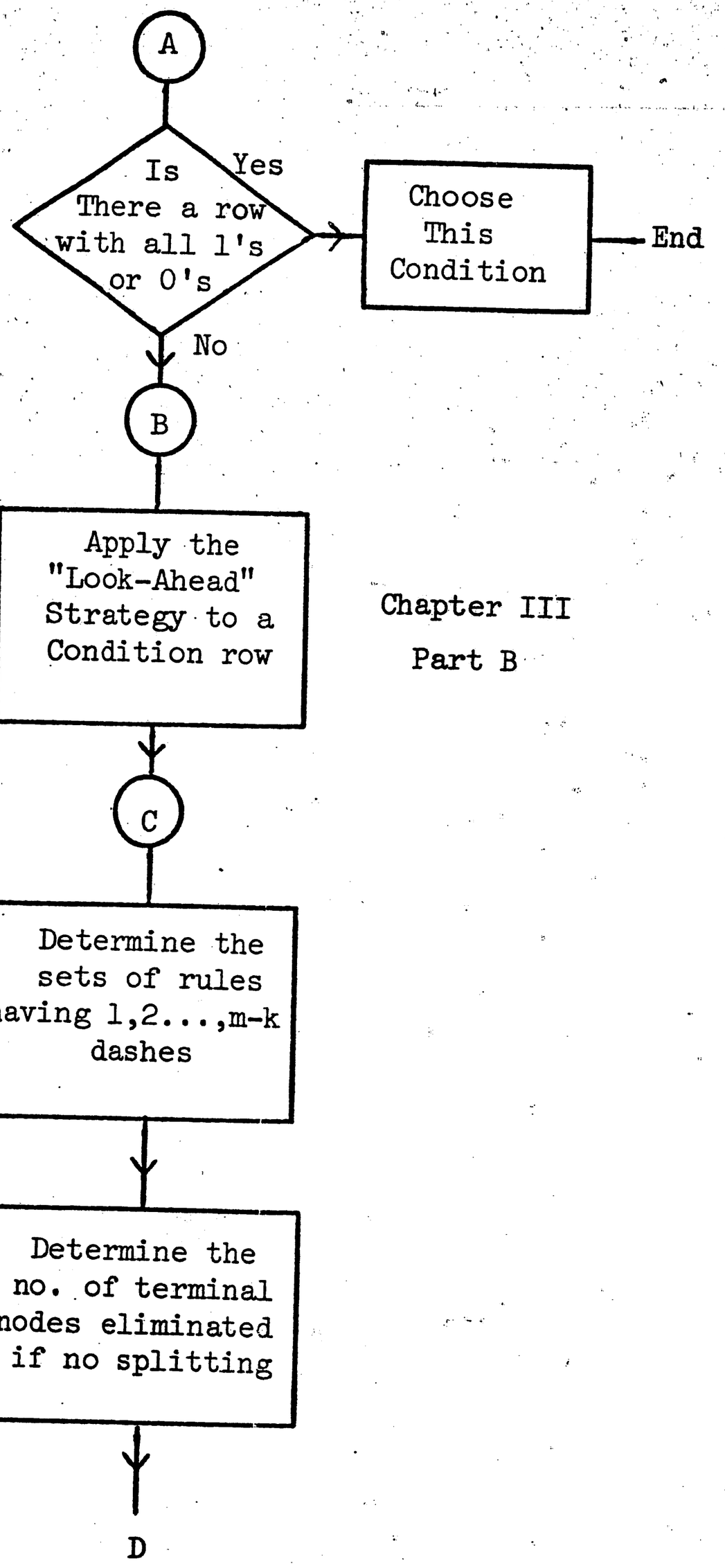
$$MX_{kj(h)} = \text{Max} \left\{ \left[MX'_{kj(h-1)} - (2^{j(h)-j(h-1)} \bullet P_{j(h)}) \right] \div 2,1 \right\}$$

where

$$MX'_{kj} = 2^{j(h)-j(h-1)} \bullet 2\, MX_{kj(h)}$$

Hence $MX_{kj(h)}$ is the preliminary maximum number of node sets of order j(h-1) that can be eliminated due to j(h-1) dashes in rules corresponding to either the zero's or one's group of a particular condition in consideration for some node at level k. The expression $2^{j(h-1)-j(h-1)}$ is the conversion factor for expressing node sets of order j(h) as an equivalent number of node sets of order j(h-1). $MX'_{kj(h-1)}$ expresses the equivalent number of node sets of order j(h-1) that are possible, based on the number of node sets of order j(h), that were available ($MX_{kj(h)}$). Finally, $MX_{kj(h-1)}$ is found by modifying $MX'_{kj(h-1)}$ by the equivalent number of nodes of order j(h-1) that were eliminated due to rules containing j(h) dashes.

The minimum storage algorithm utilizes the expressions that have been discussed to determine a condition to test when presented with a table (sub-table). A flow chart for the algorithm is presented in Figure 3.2.9. The algorithm itself is followed by example calculations to determine the root node of a decision tree.



Chapter III

Part B

FIGURE 3.2.9

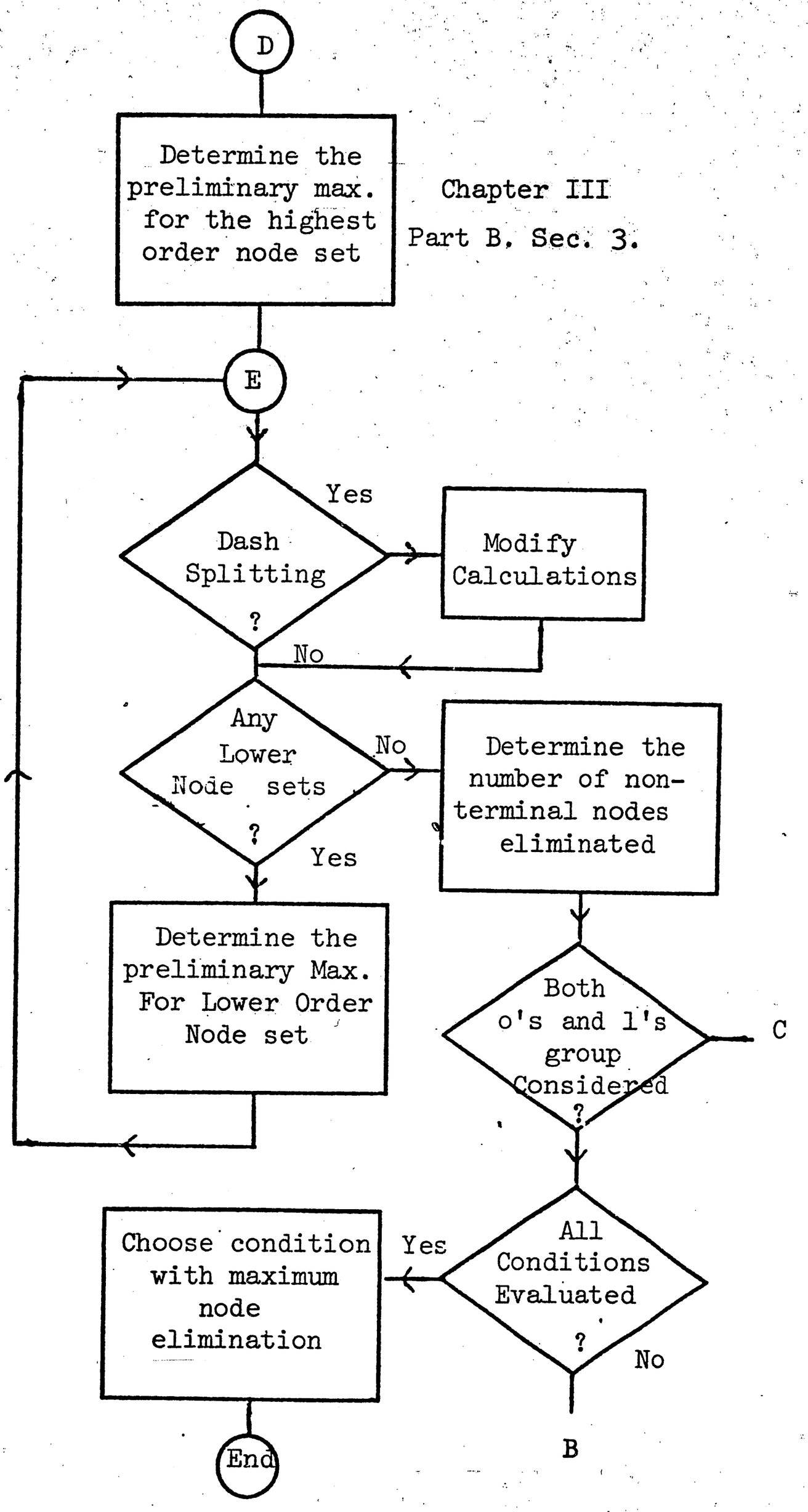


FIGURE 3.2.9 (CONTINUED)

D. The Minimum Storage Algorithm

Let $n_{(i)rk}$ = the number of columns (rules) having at least one dash for the r's group of condition (i), not including the dashes, if any, in condition (i). The subscript, k, is the level number.

Let $ND_{(j)}$ = the number of dashes in the j^{th} column of the r's group of condition (i) in consideration at level k. ($j = 1,2, \ldots, n_{(i)rk}$).

The following steps should be followed to determine the condition for each node in the creation of a decision tree. The condition being chosen from the nodes' associated sub-table at level k.

A. Choose a condition that has all ones or all zeros.

B. If A cannot be satisfied

PERFORM C. For $i = 1,2, \ldots, m-k+1$

GO TO E.

C. COMPUTE $$E_{(i)k} = \sum_{r=0}^{1} \left[\sum_{j=1}^{CR_{(i)r}} (2^{m-j-k}) - CR_{(i)r} \right]$$

PERFORM D. For r = 0,1

D. Step 1.) COMPUTE

$$\Phi_{(i)rk} = 2^{ND_{(1)}} + 2^{ND_{(2)}} + \ldots\ldots + 2^{ND_{[n_{(i)rk}]}}$$

re-grouping gives:

$$\Phi_{(i)rk} = P_1 \bullet 2^1 + P_2 \bullet 2^2 + \ldots + P_{m-k} \bullet 2^{m-k} = \sum_{t=1}^{m-k} (P_t \bullet 2^t)$$

Where P_t is the number of columns for the r's group of condition (i) at level k that have t dashes.

Step 2.) COMPUTE h = number of $P_j \neq 0$ ($j = 1,2\ldots,m-k$)

$$\text{i.e., } h = \sum_{j=1}^{m-k} i \qquad \text{where } i = \begin{cases} 1 \text{ if } P_j \neq 0 \\ 0 \text{ if } P_j = 0 \end{cases}$$

Since j(h) was defined to be the particular value of j that is the largest for $P_j \neq 0$, j(h-1) the second largest, ..., j(1) the smallest

COMPUTE

$$MX_{kj(h)} = \max\left\{\left[M_{kj(h)} \cdot 2^{m-k-j(h)} \div 2\right], 1\right\}$$

where

$$M_{kj(h)} = \begin{cases} 1 - \left[\sum_{j=1}^{CR_{(i)r}} 1/2^j\right] & \text{For } CR_{(i)r} \neq 0 \\ 1 & \text{For } CR_{(i)r} = 0 \end{cases}$$

PERFORM Step 3. For z = 0,1,2, ..., h-1.

Step 3.) IF $P_{j(h-z)} \leq MX_{kj(h-z)}$

COMPUTE

$$MX_{kj(h-z-1)} = \max\left\{\left[MX'_{kj(h-z-1)} - (2^{j(h-z)-j(h-z-1)} \cdot P_{j(h-z)})\right] \div 2, 1\right\}$$

where

$$MX'_{kj(h-z-1)} = 2^{j(h-z)-j(h-z-1)} \cdot 2MX_{kj(h-z)}$$

ELSE PERFORM Step 5.

Step 4.) COMPUTE $N_{(i)r} = \Phi_{(i)rk} - n_{(i)rk}$

GO TO Step 6, if r=1, ELSE GO TO Step 8.

Step 5.) Look among the $P_{j(h-z)}$ columns (rules) that have j(h-z) dashes for unique pairs of columns whose j(h-z) dashes are in the same condition row. Count the number of unique pairs of columns that differ in only one row and call the total $C_{j(h-z)}$.

COMPUTE $P'_{j(h-z)} = MX_{kj(h-z)} + C_{j(h-z)}$

RE-COMPUTE $\Phi_{(i)rk} = P_1 \cdot 2^1 + \ldots + P'_{j(h-z)} \cdot 2^{j(h-z)} + \ldots + P_{m-k} \cdot 2^{m-k}$

RE-COMPUTE $n_{(i)rk} = n_{(i)rk} - (P_{j(h-z)} - P'_{j(h-z)})$

COMPUTE $MX_{kj(h-z)} = \max \left\{ \left[MX'_{kj(h-z-1)} - (2^{j(h-z)} - 2^{j(h-z-1)} \cdot P'_{j(h-z)}) \right] \div 2,1 \right\}$

where MX' is the same as in Step 3.

COMPUTE $P_{j(h-z)} = P'_{j(h-z)}$

Step 6.) COMPUTE $D_{(i)k} = \sum_{r=0}^{l} N_{(i)r}$

Step 7.) COMPUTE $S_{(i)k} = E_{(i)k} + D_{(i)k}$

Step 8.) CONTINUE

E. Choose condition (i) such that $S_{(i)k}$ is maximized.

END

The algorithm is now applied to the following table to determine the root node, in order to illustrate the computations.

	R1	R2	R3	R4	R5	R6	ELSE
C1	-	-	0	1	0	0	
C2	1	1	-	0	-	1	
C3	0	1	-	-	0	1	
C4	0	-	0	-	1	1	
C5	0	0	1	-	-	1	
C6	-	-	0	0	0	-	
	A1	A2	A3	A4	A5	A6	AE

Calculations for C1

Step 1. (r = 0) $\Phi_{(1)0\ 1} = 2^1 + 2^2 + 2^2 + 2^2 + 2^1 = 16$

$n_{(1)0\ 1} = 5$; $P_1 = 2, P_2 = 3$

Step 2. (r = 0) $j(2) = 2, j(1) = 1$ where $h=2$ $M_{12} = 1, MX_{12} = 4$

Step 3. (r = 0) $P_2 = 3 < 4 = MX$

$MX'_{11} = 16$ $MX_{11} = [16 - (2^{2-1} \cdot 3)] \div 2 = 5$

$P_1 = 2 < 5 = MX_{11}$

Step 4. (r = 0) $N_{(1)0} = 16 - 5 = 11$

Step 1. (r = 1) $\Phi_{(1)11} = 2^1 + 2^2 + 2^3 = 14$

$n_{(1)11} = 3$; $P_1 = 1, P_2 = 1, P_3 = 1$

Step 2. (r = 1) $j(3) = 3, j(2) = 2, J(1) = 1$ where $h=3$

$M_{13} = 1, MX_{13} = 2$

Step 3. (r = 1) $P_3 = 1 < 2 = MX_{13}$

$MX'_{12} = 8$ $MX_{12} = [8 - (2^{3-2} \cdot 1)] \div 2 = 3$

$P_2 = 1 < 3 = MX_{12}$

$MX'_{11} = 12$ $MX_{11} = 12 - (2^{2-1} \cdot 1) \div 2 = 5$

$P_1 = 1 < 5 = MX_{11}$

Step 4. (r = 1) $N_{(1)1} = 14 - 3 = 11$

Step 6. (r = 1) $D_{(1)1} = 11 + 11 = 22$

$E_{(1)1} + D_{(1)1} = 0 + 22 = 22$

Calculations for C2

Step 1. (r = 0) $\Phi_{(2)0\ 1} = 2^1 + 2^3 + 2^1 = 12$

$n_{(2)0\ 1} = 3$; $P_1 = 2$, $P_3 = 1$

Step 2. (r = 0) j(2) = 3, j(1) = 1 where h=2 $M_{13} = 1/2$, $MX_{13} = 1$

Step 3. (r = 0) $P_3 = 1 = MX_{13}$

$MX'_{11} = 8$ $MX_{11} = [8 - (2^{3-1} \bullet 1)] \div 2 = 2$

$P_1 = 2 = MX_{11}$

Step 4. (r = 0) $N_{(2)0} = 12 - 3 = 9$

Step 1. (r = 1) $\Phi_{(2)11} = 2^2 + 2^3 + 2^1 + 2^1 + 2^1 = 18$

$n_{(2)11} = 5$; $P_1 = 3$, $P_2 = 1$, $P_3 = 1$

Step 2. (r= 1) j(3) = 3, j(2) = 2, j( ) = 1 where h=3 $M_{13} = 1$, $MX_{13} = 2$

Step 3. (r = 1) $P_3 = 1 < 2 = MX_{13}$

$MX'_{12} = 8$ $MX_{12} = [8 - (2^{2-1} \bullet 1)] \div 2 = 3$

$P_2 = 1 < 3 = MX_{12}$ $MX'_{11} = 12$ $MX_{11} =$

$[12 - (2^{2-1}\ 1)] \div 2 = 5$ $P_1 = 3 < 5 = MX_{11}$

Step 4. (r = 1) $N_{(2)1} = 18 - 5 = 13$

Step 6. (r = 1) $D_{(2)1} = 9 + 13 = 22$, $E_{(2)1} + D_{(2)1} = 15 + 22 = 37$

Calculations for C3

Step 1. (r = 0) $\Phi_{(3)0\ 1} = 2^2 + 2^1 + 2^2 + 2^2 = 14$

$n_{(3)0\ 1} = 4$; $P_1 = 2$, $P_2 = 3$

Step 2. (r = 0) $j(2) = 2, j(1) = 1$ where $h=2$ $M_{12} = 1$, $MX_{12} = 4$

Step 3. (r = 0) $P_2 = 3 < 4 = MX_{12}$

$MX'_{11} = 16$ $MX_{11} = [16 - (2^{2-1} \cdot 3)] \div 2 = 5$

$P_1 = 2 < 5 = MX_{11}$

Step 4. (r = 0) $N_{(3)0} = 14 - 4 = 10$

Step 1. (r = 1) $\Phi_{(3)11} = 2^3 + 2^1 + 2^2 + 2^1 = 16$

$n_{(3)11} = 4$; $P_1 = 2$, $P_2 = 1$, $P_3 = 1$

Step 2. (r = 1) $j(3) = 3$, $j(2) = 2$, $j(1) = 1$ where $h=3$

$M_{13} = 1$, $MX_{13} = 2$

Step 3. (r = 1) $P_3 = 1 < 2 = MX_{13}$

$MX'_{12} = 8$ $MX_{12} = [8 - (2^{3-2} \cdot 1)] \div 2 = 3$

$P_2 = 1 < 3 = MX_{12}$

$MX'_{11} = 12$ $MX_{11} = [12 - (2^{2-1} \cdot 1)] \div 2 = 5$

$P_1 = 2 < 5 = MX_{11}$

Step 4. (r = 1) $N_{(3)1} = 16 - 4 = 12$

Step 6. (r = 1) $D_{(3)1} = 10 + 12 = 22$

$E_{(3)1} + D_{(3)1} = 0 + 22 = 22$

Calculations for C4

Step 1. (r = 0) $\Phi_{(4)0\,1} = 2^2 + 2^2 + 2^2 + 2^2 = 16$

$n_{(4)0\,1} = 4$; $P_2 = 4$

Step 2. (r = 0) $j(1) = 2$ where $h=1$

$M_{12} = 1$ $MX_{12} = 4$

Step 3. (r = 0) $P_2 = 4 = MX_{12}$

Step 4. (r = 0) $N_{(4)0} = 16 - 4 = 12$

Step 1. (r = 1) $\Phi_{(4)11} = 2^2 + 2^2 + 2^2 + 2^1 = 14$

$n_{(4)} = 4$; $P_1 = 1$, $P_2 = 3$

Step 2. (r = 1) $j(2) = 2$, $j(1) = 1$ where h=2

$M_{12} = 1$, $MX_{12} = 4$

Step 3. (r = 1) $P_2 = 3 < 4 = MX_{12}$

$MX_{11} = 16$ $MX_{11} = [16 - (2^{2-1} \bullet 3)] \div 2 = 5$

$P_1 = 1 < 5 = MX_{11}$

Step 4. (r = 1) $N_{(4)1} = 14 - 4 = 10$

Step 6. (r = 1) $D_{(4)1} = 12 + 10 = 22$

$E_{(4)1} + D_{(4)1} = 0 + 22 = 22$

Calculations for C5

Step 1. (r = 0) $\Phi_{(5)0\,1} = 2^2 + 2^3 + 2^2 + 2^1 = 18$

$n_{(5)0\,1} = 4$

Step 2. (r = 0) $j(3) = 3$, $j(2) = 2$, $j(1) = 1$ where h=3

$M_{13} = 1$, $MX_{13} = 2$

Step 3. (r = 0) $P_3 = 1 < 2 = MX_{13}$

$MX'_{12} = 8$ $MX_{12} = [8 - (2^{3-1} \bullet 1)] \div 2 = 3$

$P_2 = 2 < 3 = MX_{12}$

$MX'_{11} = 12$ $MX_{11} = [12 - (2^{2-1} \bullet 2)] \div 2 = 4$

$P_1 = 1 < 4 = MX_{11}$

Step 4. (r = 0) $N_{(4)0} = 18 - 4 = 14$

Step 1. (r = 1) $\Phi_{(5)11} = 2^2 + 2^2 + 2^1 + 2^1 = 12$

$n_{(5)11} = 4$

Step 2. (r = 1) $j(2) = 2$, $j(1) = 1$ where h=2

$M_{12} = 1$, $MX_{12} = 4$

Step 3. (r = 1) $P_2 = 2 < 4 = MX_{12}$

$MX_{11} = 16$ $MX_{11} = [16 - (2^{2-1} \bullet 2)] \div 2 = 6$

$P_1 = 2 < 6 = MX_{11}$

Step 4, (r = 1) $N_{(5)1} = 12 - 4 = 8$

Step 6. (r = 1) $D_{(5)1} = 14 + 8 = 22$

$E_{(5)1} + D_{(5)1} = 0 + 22 = 22$

Calculations for C6

Step 1. (r = 0) $\Phi_{(6)0\ 1} = 2^1 + 2^2 + 2^2 + 2^3 + 2^2 = 22$

$n_{(6)0\ 1} = 5$

Step 2. (r = 0) $j(3) = 3,\ j(2) = 2,\ j(1) = 1$ where h=3

$M_{13} = 1\ ,\ MX_{13} = 2$

Step 3. (r = 0) $P_3 = 1 < 2 = MX_{13}$

$MX'_{12} = 8 \quad MX_{12} = [8 - (2^{3-2} \bullet 1)] \div 2 = 3$

$P_2 = 3 = MX_{12}$

$MX'_{11} = 12 \quad MX_{11} = [12 - (2^{2-1} \bullet 3)] \div 2 = 3$

$P_3 = 1 < 3 = MX_{11}$

Step 4. (r = 0) $N_{(6)0} = 22 - 5 = 17$

Step 1. (r = 1) $\Phi_{(6)11} = 2^1 + 2^2 = 6$

$n_{(6)11} = 2$

Step 2. (r = 1) $j(2) = 2,\ j(1) = 1$ where h = 2

$M_{12} = 1/2 \quad MX_{12} = 2$

Step 3. (r = 1) $P_2 = 1 < 2 = MX_{12}$

$MX'_{11} = 8 \quad MX_{11} = [8 - (2^{2-1} \bullet 1)] \div 2 = 2$

$P_1 = 1 < 3 = MX_{11}$

Step 4. (r = 1) $N_{(6)1} = 6 - 2 = 4$

Step 6. (r = 1) $D_{(6)1} = 17 + 4 = 21$

$E_{(6)1} + D_{(6)1} = 15 + 21 = 36$

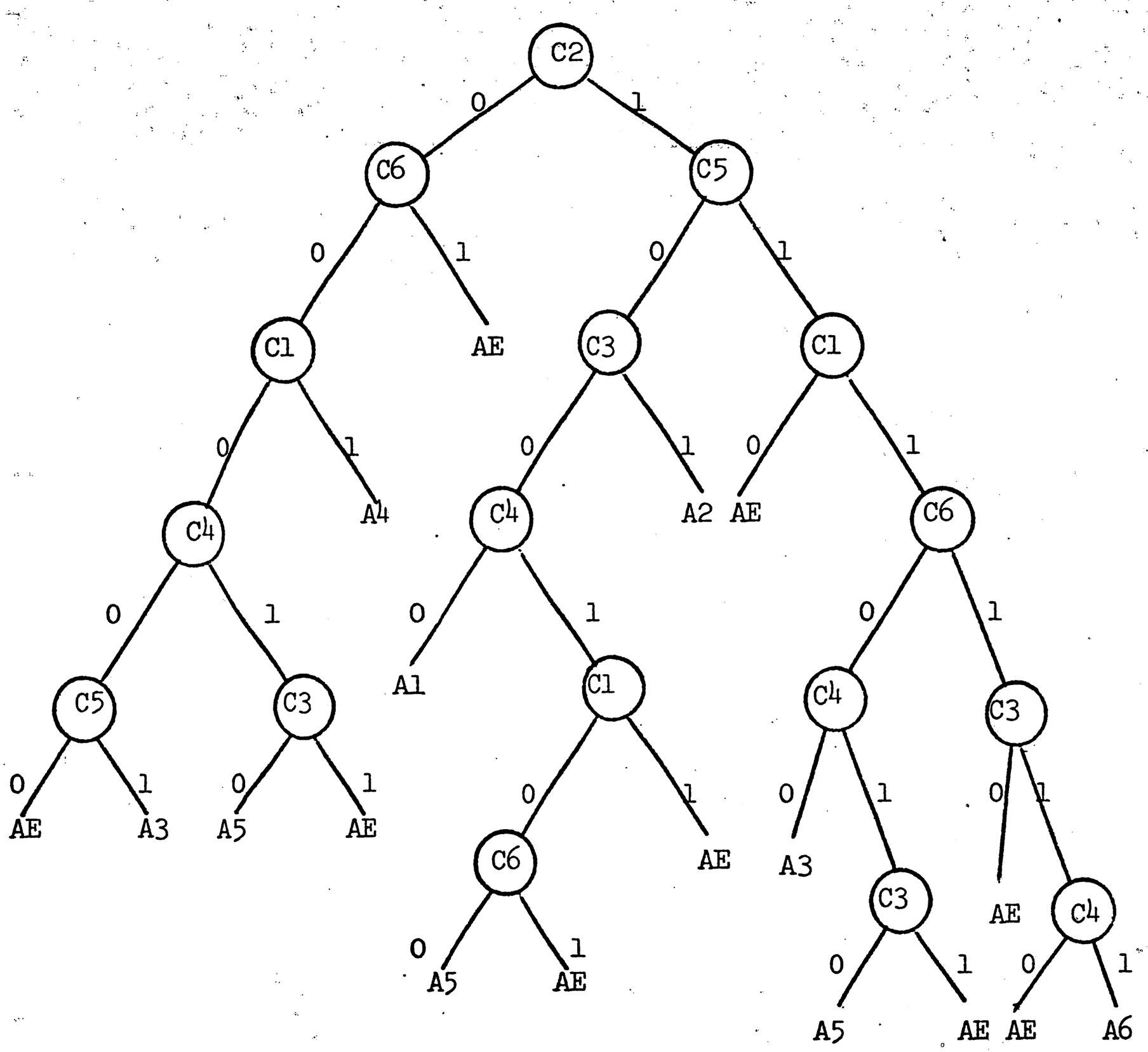

C2
0
1
C6
C5
0
1
0
1
C1
AE
C3
C1
0
1
0
1
0
1
C4
A4
C4
A2
AE
C6
0
1
0
1
0
1
C5
C3
A1
C1
0
1
0
1
0
1
AE
A3
A5
AE
C6
AE
0
1
0
1
A5
AE
C4
C3
0
1
0
1
A3
C3
AE
C4
0
1
0
1
A5
AE
AE
A6

FIGURE 3.2.10

Condition C2 has the maximum value for the expression $D_{(i)k} + E_{(i)k}$ and becomes the root node of the decision tree. The recursive application of the algorithm to the preceding table results in the decision tree in Figure 3.2.10.

E. Examples and Comparisons

1. Failure of Pollack's Algorithm

Consider the following table:

	R1	R2	R3	R4	R5	ELSE
C1	0	0	1	1	1	
C2	1	1	-	1	0	
C3	0	0	0	1	1	
C4	0	1	0	0	0	
	A1	A2	A3	A4	A5	AE

In determining the root node of the decision tree, the proposed algorithm gives the following:

C1. $D_{(1)1} + E_{(1)1} = 8$ *max. value

C2. $D_{(2)1} + E_{(2)1} = 4$

C3. $D_{(3)1} + E_{(3)1} = 5$

C4. $D_{(4)1} + E_{(4)1} = 5$

Pollack's [14] algorithm, when applied to the table, chooses C4 for the root node. His dash count and delta values are as follows:

	Dash Count	Delta
C1.	0	2
C2.	2	N/A
C3	0	2
C4	0	4 *max delta

Recursive application of both algorithms results in the decision trees in Figure 3.2.11.

2. Inconsistency of Pollacks Algorithm

Consider the following table:

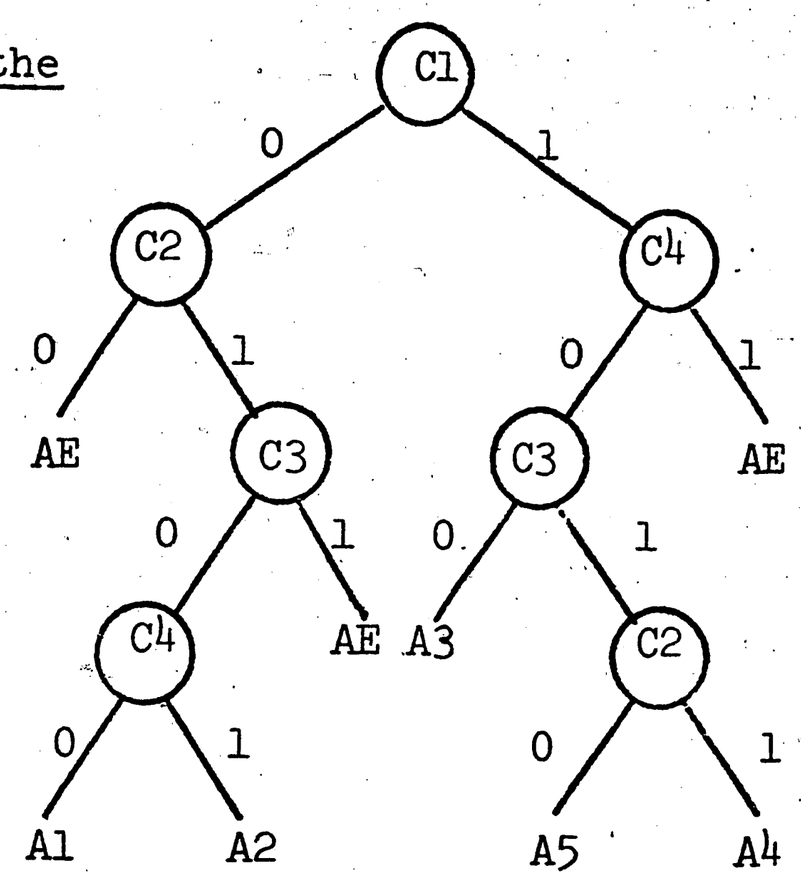
	R1	R2	R3	R4	R5	ELSE
C1	1	1	1	1	1	
C2	0	0	1	1	1	
C3	1	1	0	0	1	
C4	0	1	1	0	0	
	A1	A2	A3	A4	A5	AE

Both Pollack's algorithm and the proposed algorithm would choose C1 for the root node. The resulting sub-table off the one's branch of C1 is:

	R1	R2	R3	R4	R5	ELSE
C2	0	0	1	1	1	
C3	1	1	0	0	1	
C4	0	1	1	0	0	
	A1	A2	A3	A4	A5	AE

Pollack's algorithm indicates a tie between all three conditions C2, C3 and C4 for the next node (i.e., delta = 1 for all three). The proposed

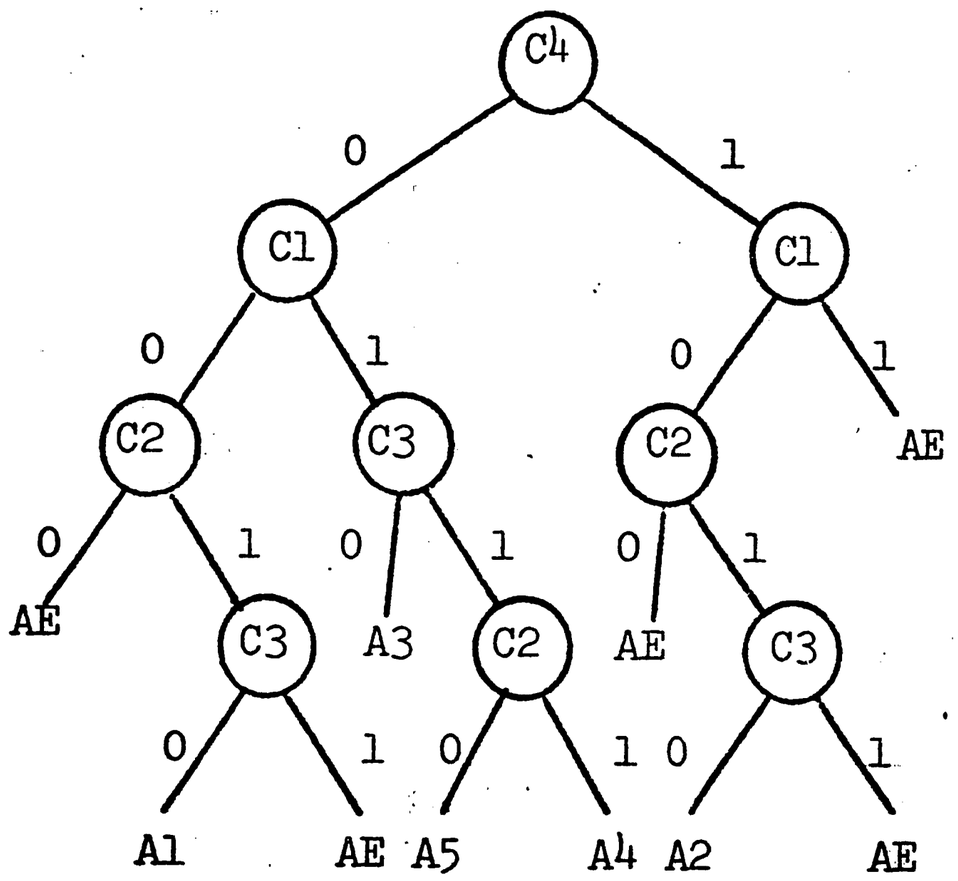
Tree generated by the proposed algorithm



7 non-terminal nodes*

*The object program would require a smaller amount of storage than one derived from Figure 3.2.11

Tree generated by Pollack's algorithm



9 non-terminal nodes

FIGURE 3.2.11

algorithm indicates that only C2 and C3 are equally good choices and would be better than C4 (i.e., $D_{(2)2} + E_{(2)2} = 1$; $D_{(3)2} + E_{(3)2} = 1$; $D_{(4)2} + E_{(4)2} = 0$). The three trees are shown in Figure 3.2.12 and illustrate the inconsistency of Pollack's algorithm. If C4 is chosen off the one's branch of C1, a decision tree with eight non-terminal nodes results, as opposed to seven when either C1 or C3 are chosen.

3. Consistency of Proposed Algorithm

Consider the following table:

	R1	R2	R3	R4	R5	ELSE
C1	-	0	0	1	1	
C2	0	-	0	1	0	
C3	0	1	-	0	1	
C4	0	0	1	-	-	
	A1	A2	A3	A4	A5	AE

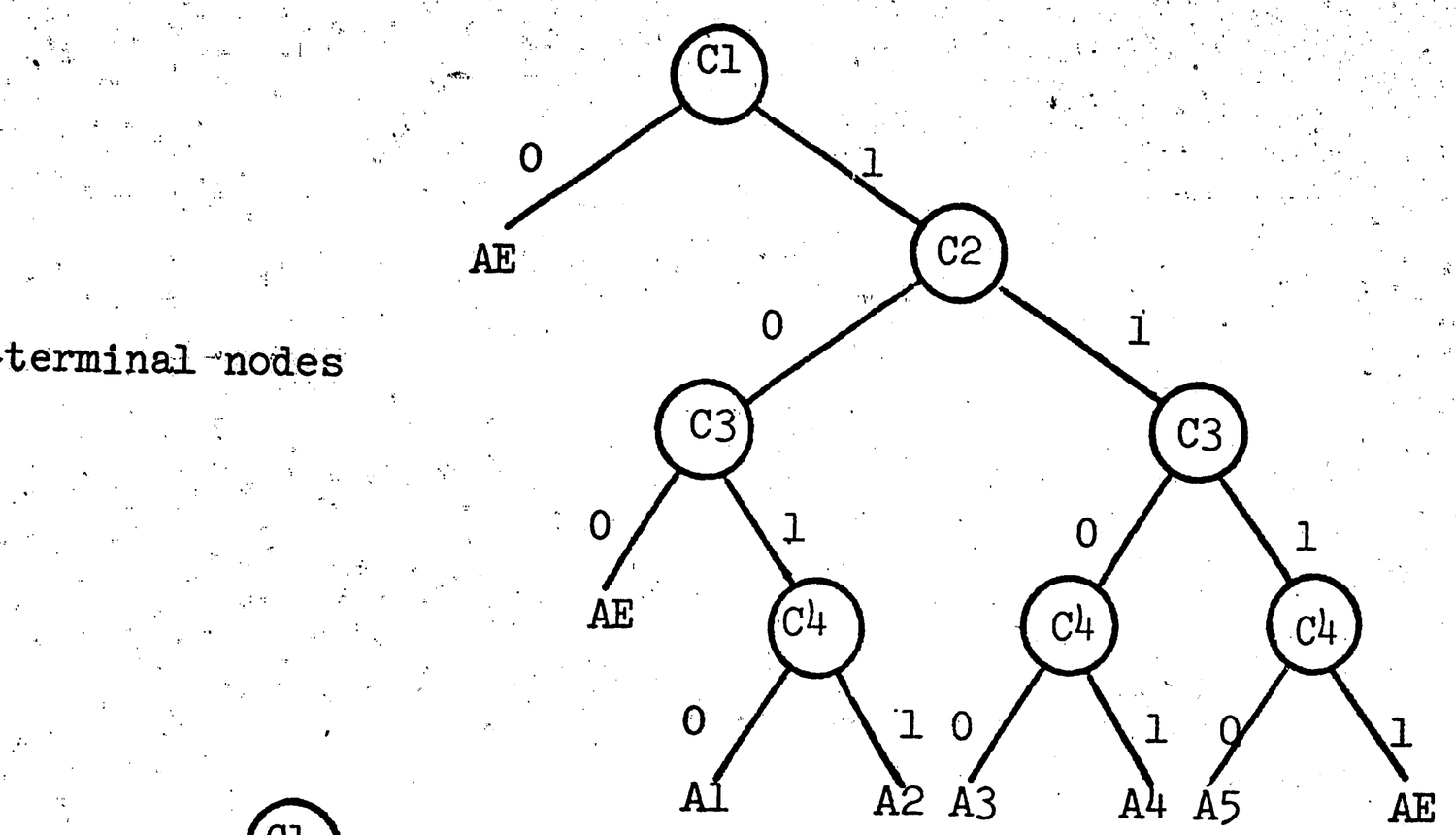
The proposed algorithm gives these values:

$D_{(1)1} + E_{(1)1} = 4$; $D_{(2)1}\ E_{(2)1} = 3$; $D_{(3)1} + E_{(3)1} = 4$; $D_{(4)1} + E_{(4)1} = 3$.

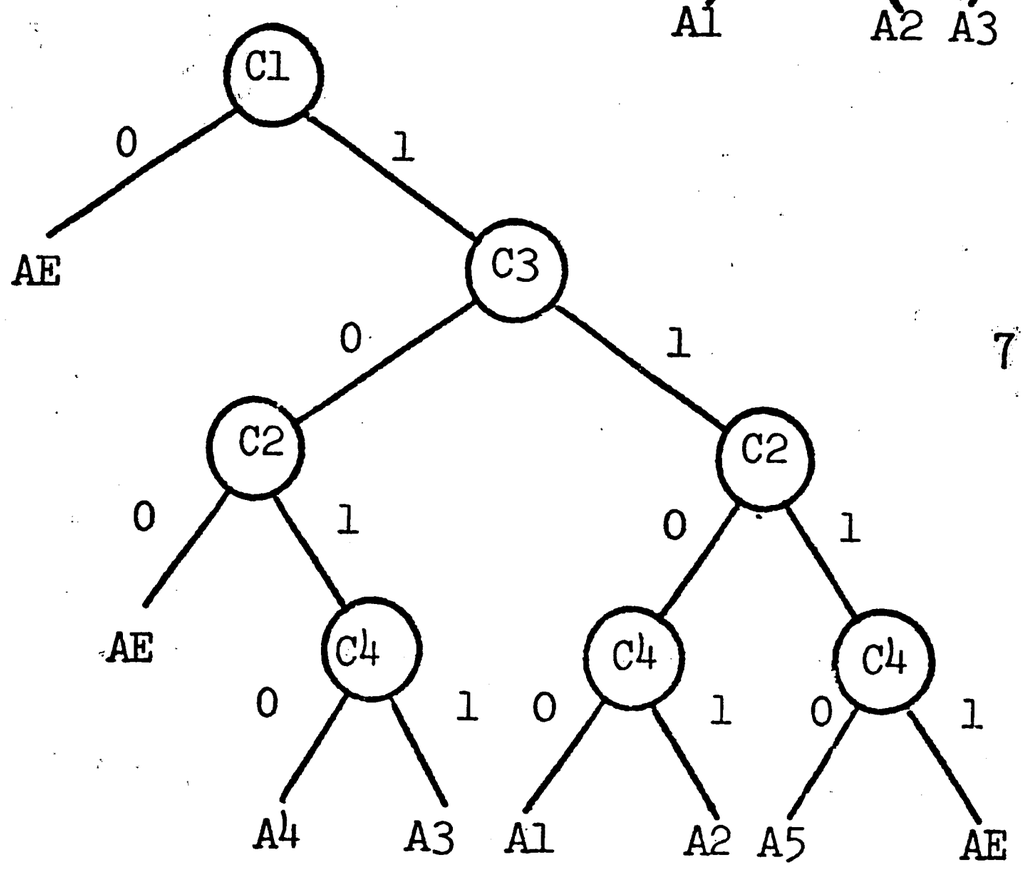
Thus, either C1 or C3 would be good choices for the root node in the decision tree. Pollack's algorithm gives:

	Dash Count	Delta	
C1	2	0	
C2	2	4	*max
C3	2	0	
C4	4	N/A	

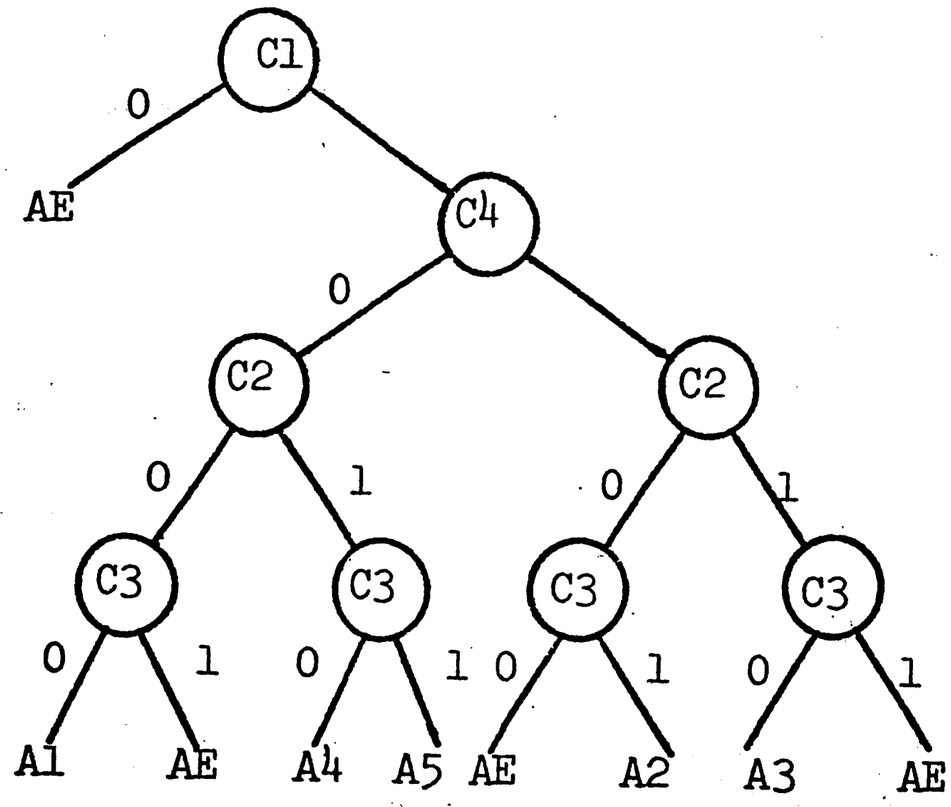
The indication here is that C2 should be chosen for the root node. The decision trees generated from the recursive application of both algorithms



7 non-terminal nodes



7 non-terminal nodes



8 non-terminal nodes

FIGURE 3.2.12

are seen in Figure 3.2.13. It should also be noted that Presses[18] algorithm would consider all four conditions as equally good choices.

4. Incompleteness of Pollack's Algorithm and Press' Algorithm

Consider the following table:

	R1	R2	R3	R4	ELSE
C1	1	0	1	0	
C2	1	-	0	1	
C3	0	1	0	0	
C4	1	1	1	0	
	A1	A2	A3	A4	AE

The calculations for the root node using the proposed algorithm, Pollack's algorithm and Presses' algorithm are presented.

Proposed

$D_{(1)1} + E_{(1)1} = 1 + 4 = 5$ *

$D_{(2)1} + E_{(2)1} = 0 + 3 = 3$

$D_{(3)1} + E_{(3)1} = 1 + 4 = 5$ *

$D_{(4)1} + E_{(4)1} = 1 + 4 = 5$ *

*tie between C1, C2, and C4

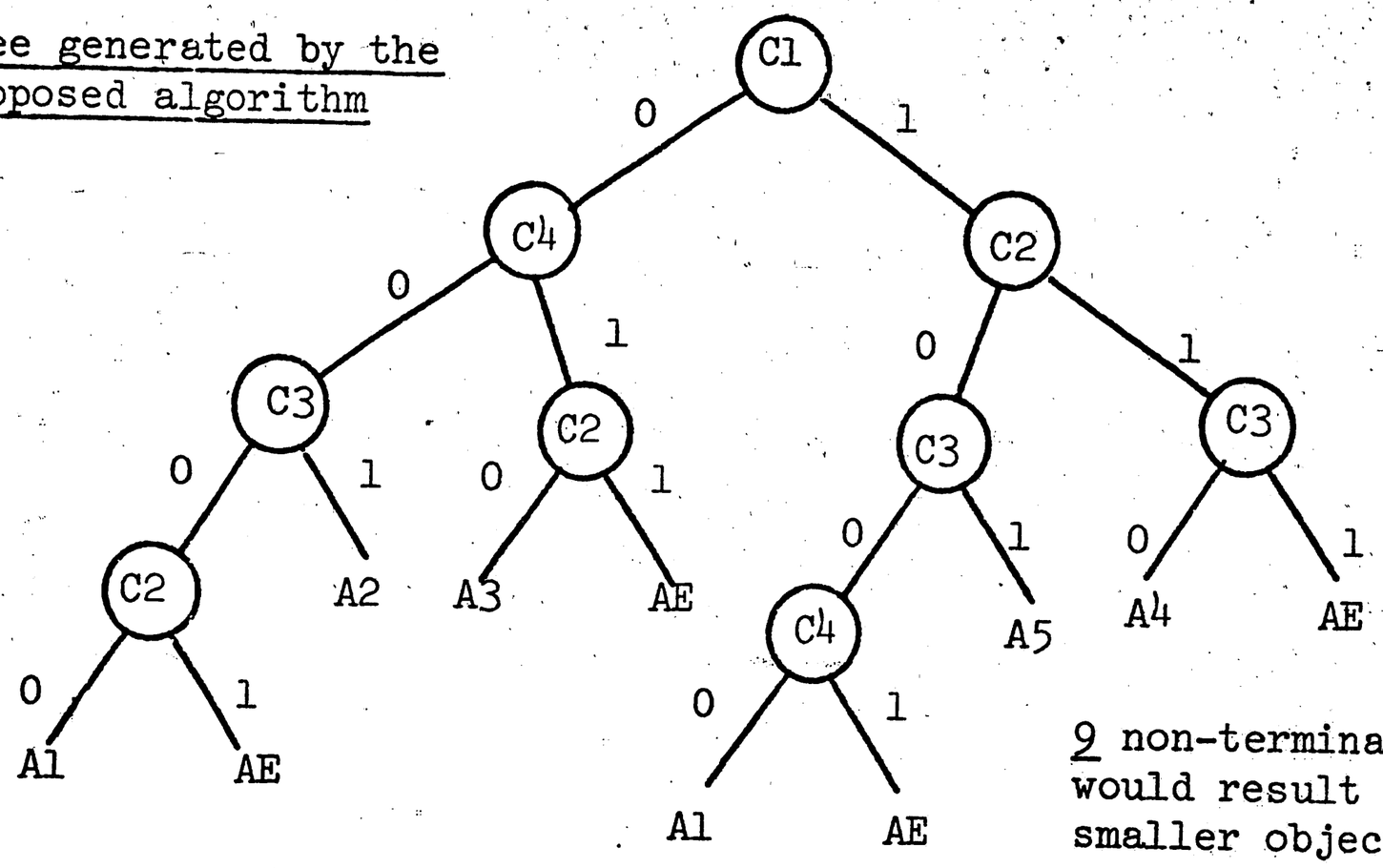
Pollack

	Dash Count	Delta
C1	0	1
C2	2	N/A
C3	0	1
C4	0	3 *max

Press

C1's Count is 4 *max
C2's " " " " 4 *
C3's " " " " 2
C4's " " " " 3

Tree generated by the proposed algorithm



9 non-terminal nodes would result in the smaller object program

Tree generated by Pollack's algorithm

10 non-terminal nodes

FIGURE 3.2.13

Pollack indicates that C4 is the best choice for the root node. Press indicates C1 or C4 for the node. The proposed algorithm shows that C1, C3 or C4 will be equally good choices and will be better choices than C2. Thus, alternative choices are provided by the proposed algorithm due to the more through analysis in the complete tree approach. The decision trees generated by recursive application of the proposed algorithm when C1, C2 and C4 are each used for the root node are illustrated in Figure 3.2.14.

5. Failure of Pollack's Algorithm

Consider the following table:

	R1	R2	R3	R4	R5	ELSE
C1	0	0	0	1	1	
C2	0	0	0	0	1	
C3	0	1	0	0	0	
C4	0	1	1	-	-	
	A1	A2	A3	A4	A5	AE

The proposed algorithm gives these values:

$D_{(1)1} + E_{(1)1} = 8$, $D_{(2)1} + E_{(2)1} = 6$, $D_{(3)1} + E_{(3)1} = 6$, $D_{(4)1} + E_{(4)1} = 3$.

Thus, C1 would be the best choice for the root node in the decision tree.

Pollack's algorithm gives:

C1	0	1
C2	0	3
C3	0	5 *max delta
C4	4	N/A

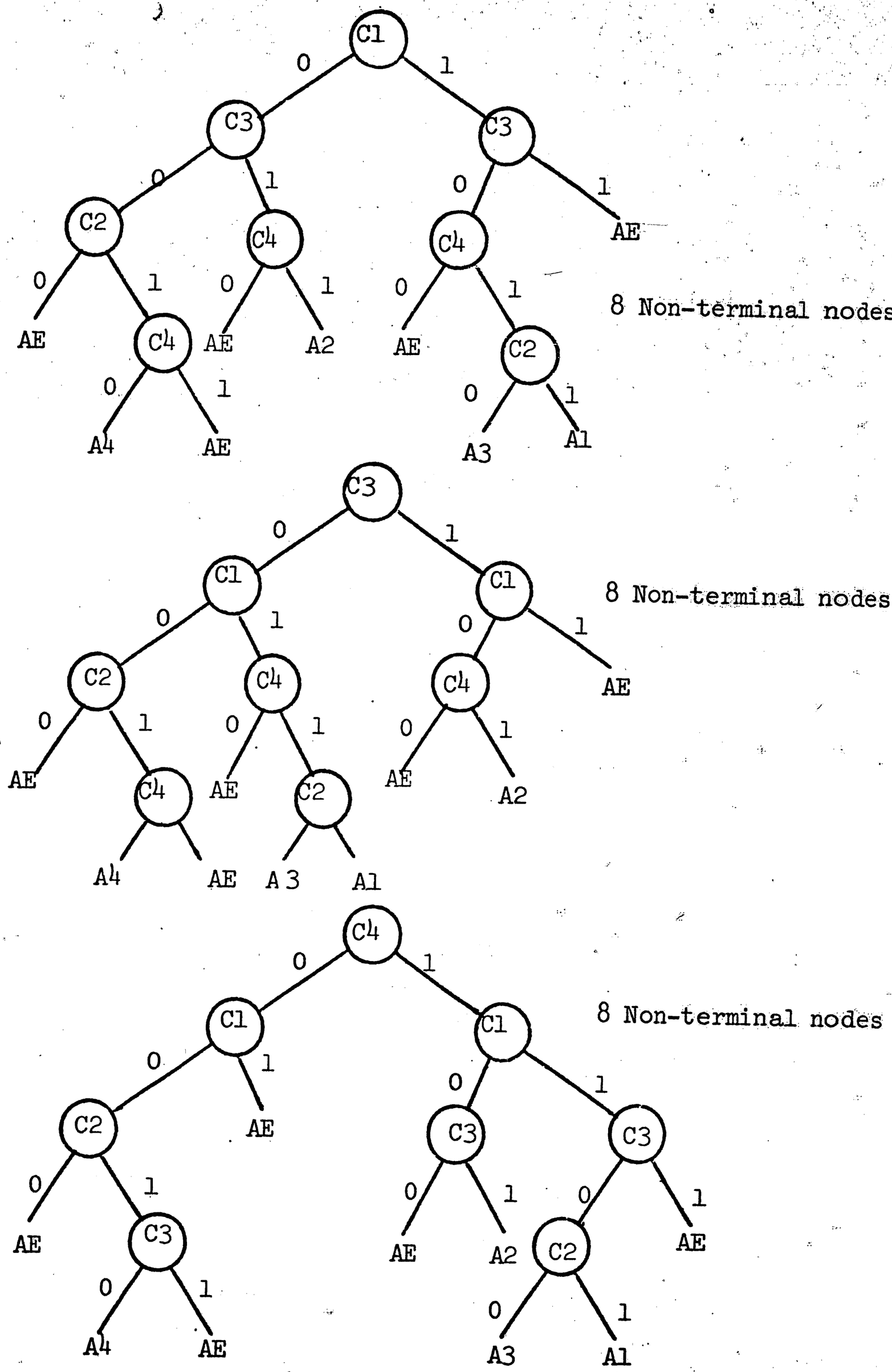
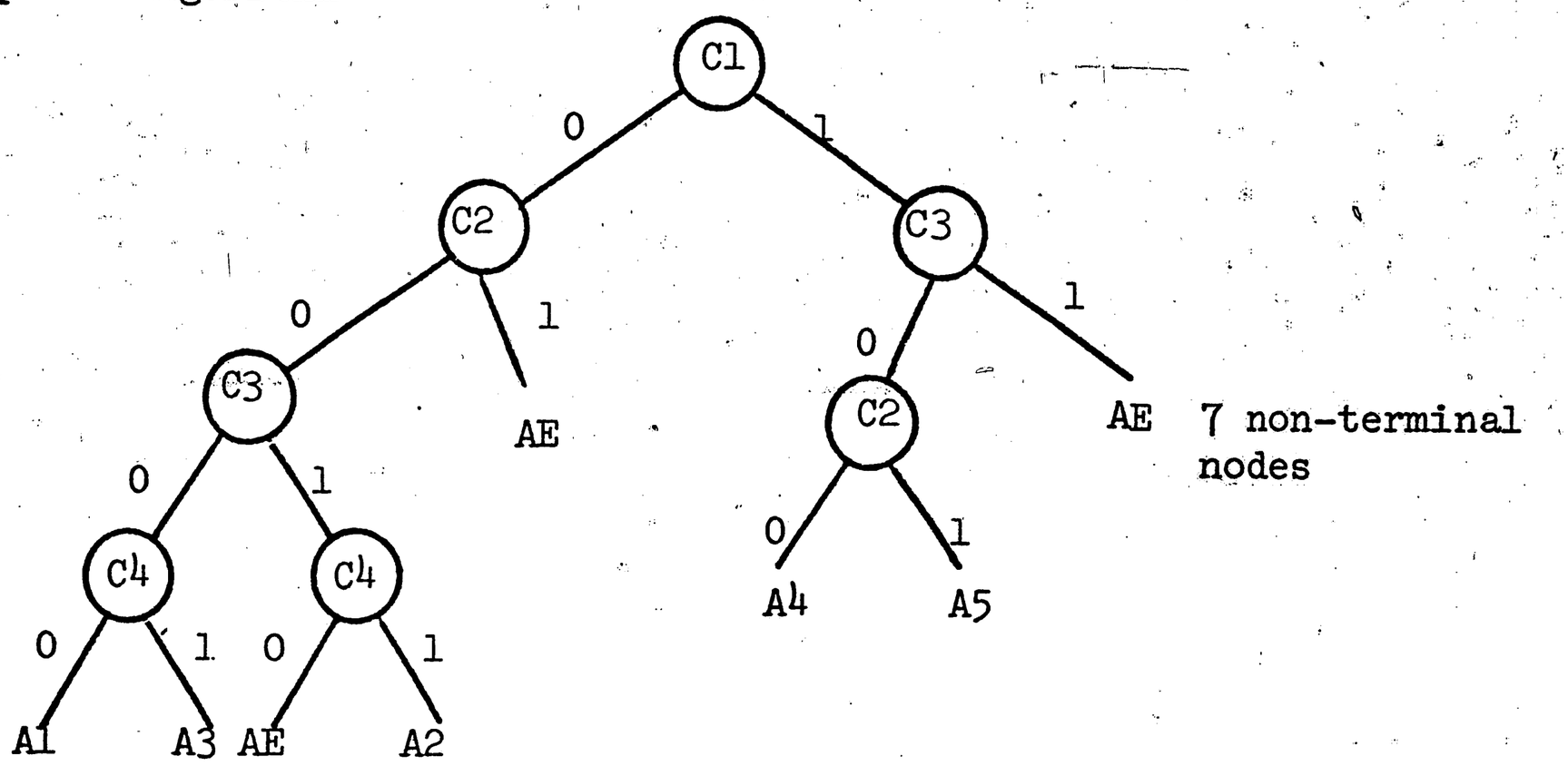

C1
0
1
C3
C3
0
1
0
1
C2
C4
C4
AE
0
1
0
1
0
1
AE
C4
AE
A2
AE
C2
0
1
0
1
A4
AE
A3
A1
C3
0
1
C1
C1
0
1
0
1
C2
C4
C4
AE
0
1
0
1
0
1
AE
C4
AE
C2
AE
A2
0
1
0
1
A4
AE
A3
A1
C4
0
1
C1
C1
0
1
0
1
C2
AE
C3
C3
0
1
0
1
0
1
AE
C3
AE
A2
C2
0
1
0
1
A4
AE
A3
A1
8 Non-terminal nodes
8 Non-terminal nodes
8 Non-terminal nodes

FIGURE 3.2.14

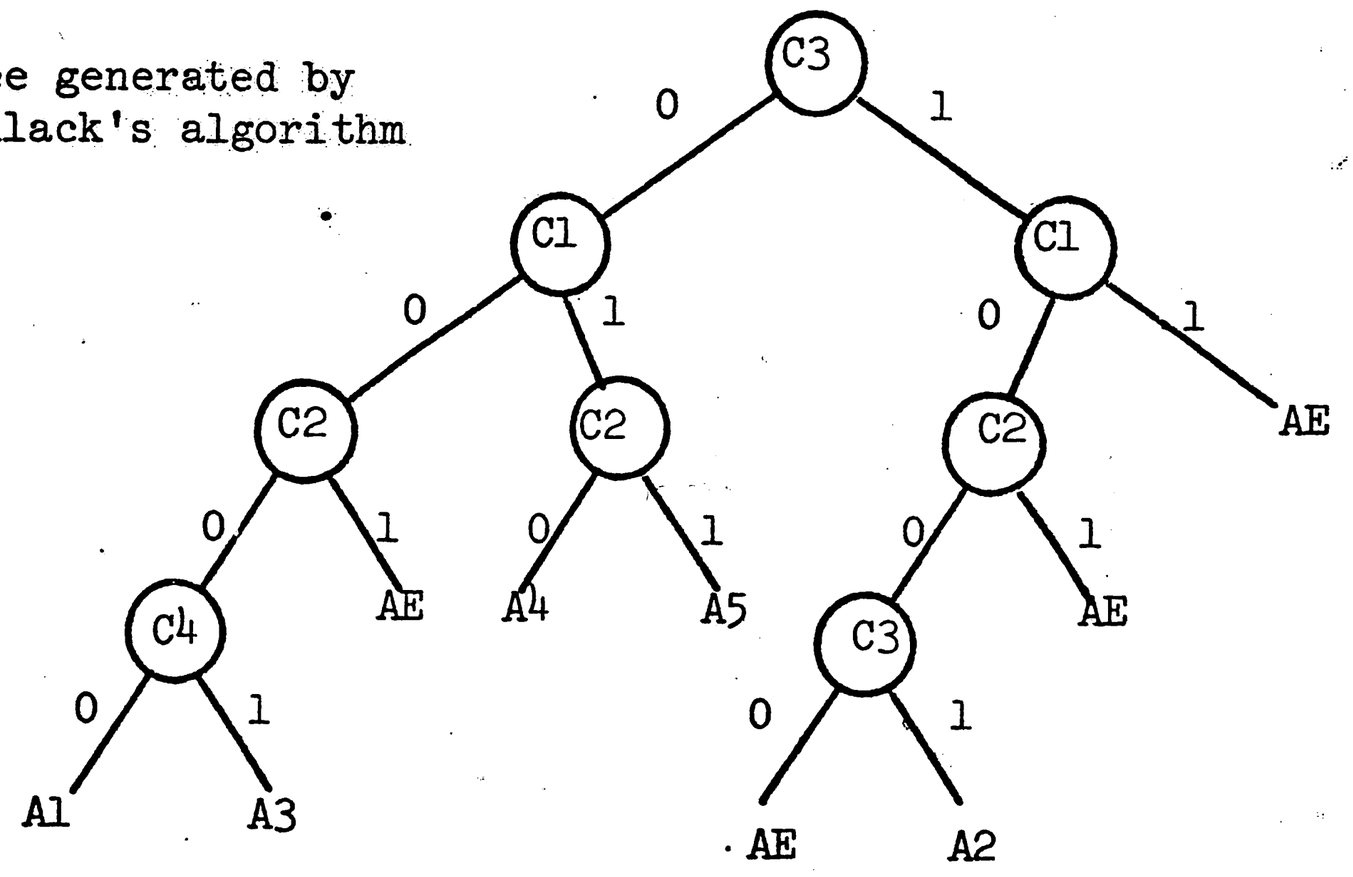
The condition C3 is indicated to be the best choice for the root node. This, however, is not the case and can be seen in Figure 3.2.15, where the decision trees are presented resulting from recursive application of both algorithms.

Tree generated by the proposed algorithm



7 non-terminal nodes

Tree generated by Pollack's algorithm



8 non-terminal nodes

FIGURE 3.2.15

CHAPTER IV

OPTIMIZATION WITH MINIMUM RESPONSE TIME CONSIDERATIONS

A. Rule Probabilities and Condition Test Times

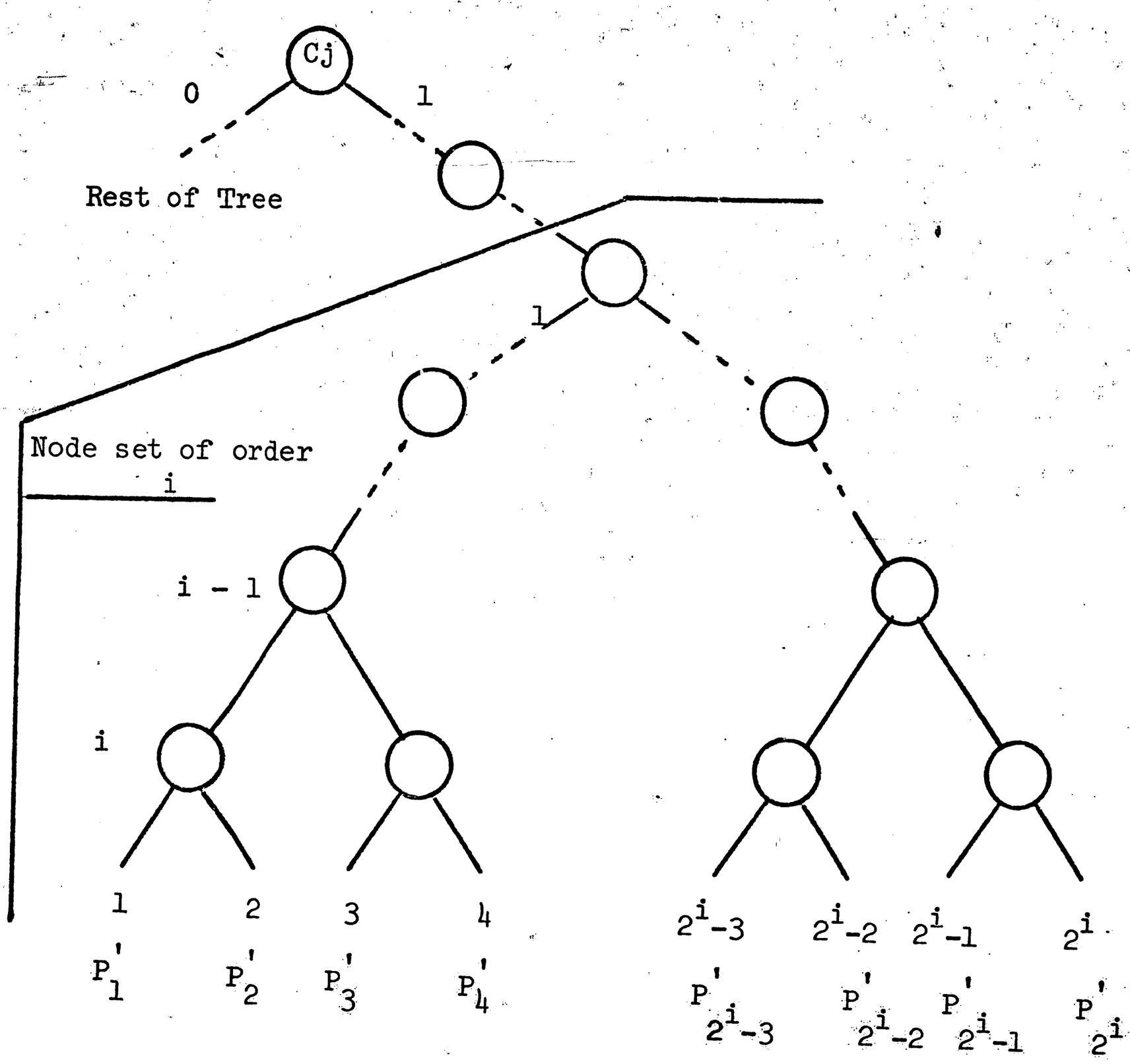
In the development of an algorithm with the goal of minimizing the response (run) time of an object program generated from a decision tree, the approach taken will be to minimize the expected time to travel from the root node to a terminal node. If each path through the decision tree (corresponding to a rule in a decision table) is equally likely and each condition requires the same test time, then the minimum response time tree would be the one with the fewest non-terminal nodes. Thus, the minimum storage algorithm (Chapter III) would accomplish both. This, however, is not the general case. Transactions may have different relative frequencies associated with them. This causes their identification with rules in a table and the execution of the proper action sets to be probabilistic. Likewise, the times associated with condition testing may vary from one condition to another. As seen in Chapter II the latter is disregarded in the existing algorithms with the exception of the algorithm by Verhelst[23]. Also the provision for an ELSE rule is not provided.

The determination of the relative frequencies of the transactions in a real-time system may be determined from a system logging tape. Consider a data collection system operating with a fixed interrupt that causes execution of a hardware driver program. This program samples input lines for arriving transactions which are then edited and passed on to other programs for processing. The transactions, including

errors, are recorded on a logging tape. A decision table has been created for use in designing the edit program. There is a need to minimize the expected execution time of the edit program in order to help maximize the remaining processing time before the next fixed interrupt. In order to associate probabilities with the rules in the decision table, the relative frequencies of the transactions are recorded over some predetermined time interval. Each transaction has a corresponding simple rule. The relative frequencies (probabilities) of the table rules are obtained by summing the frequencies of the simple rules that were combined when the table rule was formed. This also holds true for the ELSE rule.

B. Criteria of Expected Savings

In the minimum storage algorithm we were concerned with estimating the number of non-terminal nodes that could be eliminated from a complete tree due to the ELSE rule (look-ahead strategy) and the dashes in the table. Let us recall the one-to-one relationship between the order i of a node set that is eliminated from a complete tree and the number of dashes in a rule that provides for the elimination. Assume that a condition Cj is being considered for the root node of a decision tree and if Cj is chosen, there exists a partial tree where a node set of order i has been eliminated from a complete tree. This node set is eliminated due to i dashes in rule RL (Figure 4.1). Each path from the root node of the complete tree to one of the 2^i terminal nodes for the node set of order i (indicated in Figure 4.1) has i less non-terminal nodes after the elimination of the node set.


Cj
0
1
Rest of Tree
1
Node set of order
i
i - 1
i
1
2
3
4
2^i-3
2^i-2
2^i-1
2^i
P'_1
P'_2
P'_3
P'_4
P'_{2^i-3}
P'_{2^i-2}
P'_{2^i-1}
P'_{2^i}

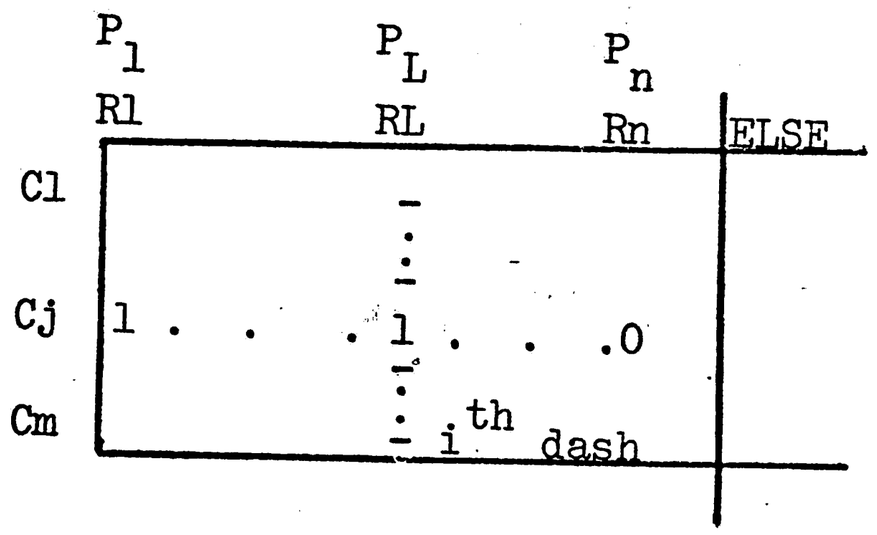

P_1
P_L
P_n
R1
RL
Rn
ELSE
C1
Cj
Cm
1 . . . 1 . . . 0
i^{th} dash

FIGURE 4.1

The i non-terminal nodes (conditions), eliminated in each path, are the conditions that have dash entries in the rule RL in the table. Now, let the probabilities of taking each of the 2^i paths and the test times associated with the i conditions eliminated in each path be:

$$P_1', P_2', \ldots, P_{2^i}' \text{ and } T_1, T_2, \ldots, T_{2^i}$$

respectively.

The expected time saved in any of the 2^i paths when the node set of order i is eliminated is:

$$P_1'(T_1 + T_2 + \ldots + T_i) + P_2'(T_1 + T_2 \ldots + T_i) + \ldots +$$

$$P_{2^i}'(T_1 + T_2 + \ldots + T_i) = (P_1' + P_2' + \ldots + P_{2^i}')(T_1 + T_2 + \ldots + T_i).$$

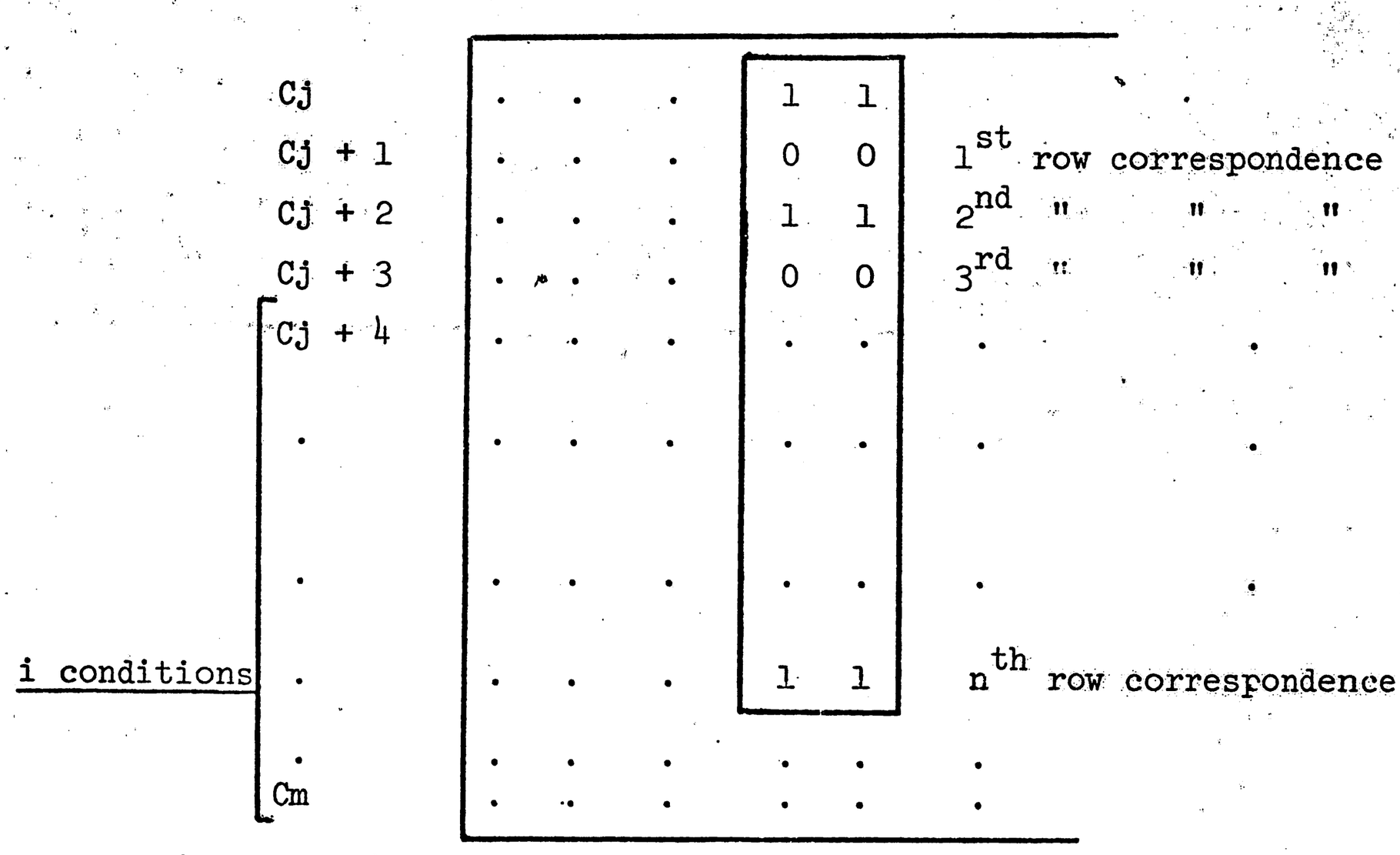
If we let $P_1' + P_2' + \ldots + P_{2^i}' = P_L$, then we can write the expected time saved as: $P_L \cdot \sum_{q=1}^{i} T_q$. Since P_L is the sum of the probabilities of the 2^i paths (simple rules), it becomes the probability associated with the table rule RL that has i dashes (Figure 4.1).

When the ELSE rule accounts for the elimination of a node set of order i, the expected savings is $2^i \cdot P_e \cdot \sum_{q=1}^{i} T_q$. Here P_e is the probability of any simple rule in the ELSE rule and it is assumed that the probability of each simple rule in the ELSE rule is the same. Otherwise, the expression would be $\sum_{j=1}^{2^i} P_{ej} \cdot \sum_{Q=1}^{i} T_q$, where P_{ej} is

the probability associated with the j^{th} simple rule in the ELSE rule. The test times T_1, T_2, ..., T_i are associated with the i conditions that are eliminated in the node set of order i. These conditions are determined from a decision table as follows: Assume that conditon Cj is being considered for the root node of a decision tree and that a node set of order i is eliminated from a complete tree due to the third row correspondence with the one's group of Cj (Figure 4.2). In other words, the third row correspondence in the "look-ahead" strategy foresees the existence of a sub-table in the creation of the decision tree that has a row of all zeros. The choosing of this condition for a non-terminal node eliminates a node set of order i, since its one's branch points to the ELSE action set. The i conditions eliminated in the node set of order i are indicated in Figure 4.2. These are the conditions that have not been involved in any previous row correspondences. Note that the method for numbering the row correspondences is sequential from the top of the table to the bottom and is strictly for convenience. Since all the conditions in a row correspondence with Cj are tested off its one's branch, the time saved due to ELSE rule eliminations is the same regardless of their order.

C. The Minimum Response Time Algorithm

By using the framework of the minimum storage algorithm to analyze dash splitting and by calculating expected time savings for node set eliminations, the minimum response time algorithm can be presented. The algorithm is recursively applied to each sub-table in the creation



(m = j+3+i)

Arbitrary Choice of the 3rd row correspondences with Cj such that a node set of order i is eliminated.

FIGURE 4.2

of a decision tree to determine the condition to test. The subscript k represents the current level in the decision tree.

A. Choose a condition row that has all ones or all zeros.

B. If A. cannot be satisfied, PERFORM C. For $i = 1, 2, \ldots, m - k + 1$.

C. COMPUTE ESAV = $\sum_{j=1}^{CR_{(i)r}} \left(2^{m-j-k} \cdot Pe \cdot \sum_{L=1}^{m-j-k} T_{(L)} \right)$

PERFORM D. For r = 0, 1

D. Step 1.

COMPUTE the matrices for $j = 1, 2, \ldots, m - k$

$$TEST_j = \begin{bmatrix} A_1 & A_2 \cdots & A_{NCOL_j} \\ P_1 & P_2 \cdots & P_{NCOL_j} \\ T_{11} & T_{12} \cdots & T_{1,NCOL_j} \\ \vdots & \vdots & \\ T_{j1} & T_{j2} \cdots & T_{j,NCOL_j} \end{bmatrix}$$

Where:

A. $NCOL_j$ is the number of columns (rules) having exactly j dashes, excluding any dashes in row i itself.

B. The a_t values are the column subscripts in the decision table for the $NCOL_j$ rules having j dashes.

C. The P_t values are the conditional probabilities

associated with the rules referred to by the a_t subscribts. If the condition row i in consideration has a dash in the rule denoted by a_t, then the probability P_t in the matrix $TEST_j$ is actually $P_i/2$. This assumes an equal split on the rule probability P_i for the cases when the dash is zero and when the dash is one.

D. T_{qt} is the time acsociated with testing the q^{th} condition having a dash in the t^{th} rule refered to by a_t. $(q = 1,j)$ $(t = 1, NCOL_j)$

Step 2.

COMPUTE h = number of $NCOL_j \neq 0$, $(j = 1,2, \ldots, m - k)$

i.e., $h = \sum_{j=1}^{m-k} i$, where $i = \begin{cases} 1 \text{ if } NCOL_j \neq 0 \\ 0 \text{ if } NCOL_j = 0 \end{cases}$

Let $j(h) = \max (j)$

$\cdot$

$\cdot$

$\cdot$

$j(1) = \min (j)$

COMPUTE $MX_{kj(h)} = \max \left\{ \left[(M_{kj(h)} \cdot 2^{m-k-j}) + 2 \right], 1 \right\}$

where $M_{kj(h)} = \begin{cases} 1 - \sum_{j=1}^{CR_{(i)r}} 1/2^j & \text{For } CR_{(i)r} \neq 0 \\ 1 & \text{For } CR_{(i)r} = 0 \end{cases}$

PERFORM Step 3. For $z = 0, 1, \ldots, h - 1$

Step 3.

If $NCOL_{j(h-z)} \leq MX_{kj(h-z)}$

COMPUTE (see next page)

$$MX_{kj(h-z-1)} = \max \left\{ \left[MX'_{kj(h-z)} - (2^{j(h-z)-j(h-z-1)} \bullet NCOL_{j(h-z)}) \div 2 \right], 1 \right\}$$

Where

$$MX'_{kj(h-z-1)} = 2^{j(h-z) - j(h-z-1)} \bullet 2MX_{kj(h-z)}$$

ELSE PERFORM Step 5.

Step 4.

$$\text{COMPUTE} \quad DSAV_{(i)r} = \sum_{L = j(1)}^{j(h)} \sum_{t = 1}^{NCOL_L} \left[\sum_{q = 1}^{L} T_{qt} \bullet P_t \right]$$

If r = 1 GO TO Step 6. ELSE GO TO Step 8.

Step 5.

Look among the $NCOL_{j(h-z)}$ columns that have j(h-z) dashes for unique pairs of columns whose j(h-z) dashes are in the same condition row. Count the number of unique pairs of columns that differ in only one row and call the total $C_{j(h-z)}$. COMPUTE $NCOL'_{j(h-z)} = MX_{kj(h-z)} + C_{j(h-z)}$

RE-COMPUTE

$$TEST_{j(h-z)} = a_1\ a_2 \cdots a_{NCOL'_{j(h-z)}}$$

$$P_1\ P_2 \cdots P_{NCOL'_{j(h-z)}}$$

$$T_{11}\ T_{12} \cdots T_{1, NCOL_{j(h-z)}}$$
$$\vdots$$
$$\vdots$$
$$T_{j(h-z)1} \cdots T_{j(h-z), NCOL_{j(h-z)}}$$

by deleting the $(NCOL_{j(h-z)} - NCOL'_{j(h-z)})$ columns whose

values $\left[\sum_{q=1}^{j(h-z)} T_{qt}\right] \cdot P_t$ are the smallest. The

$(NCOL_{j(h-z)} - NCOL'_{j(h-z)})$ columns (rules) eliminated will have their dashes split. The columns are chosen such that the expected sum of the test times of the conditions that would have been eliminated(had the dashes not been split) is minimal. This forces the dashes that are not split to have the largest possible expected time savings associated with them; thus, maximizing the expected savings calculation.

COMPUTE

$$MX_{kj(h-z-1)} = \max\left\{\left[MX'_{kj(h-z-1)} - (2^{j(h-z)-j(h-z-1)} \cdot NCOL'_{j(h-z)}) \div 2\right], 1\right\}$$

where MX' is the same as in Step 3.

COMPUTE $NCOL_{j(h-z)} = NCOL'_{j(h-z)}$

Step 6.

COMPUTE $XDSAV_{(i)k} = \sum_{r=0}^{1} DSAV_{(i)r}$

Step 7.

COMPUTE $EXSAV_{(i)k} = ESAV + XDSAV_{(i)k}$

Step 8. Continue

E. Choose the condition (i) such that $EXSAV_{(i)k}$ is maximum.

END

The algorithm is applied to the following table for condition C1 to illustrate the computations.

	P_1	P_2	P_3	P_4	P_5	P_6	PE
	.05	.1	.3	.1	.05	.2.	.2

			R1	R2	R3	R4	R5	R6	ELSE
T_1	100	C1	-	-	0	1	0	0	
T_2	50	C2	1	1	-	0	-	1	
T_3	200	C3	0	1	-	-	0	1	
T_4	25	C4	0	-	0	-	1	1	
T_5	100	C5	0	0	1	-	-	1	
T_6	100	C6	-	-	0	0	0	-	
			A1	A2	A3	A4	A5	A6	AE

Calculations For C1

Step 1. (r = 0)

$$CST_1 = \begin{bmatrix} 1 & 6 \\ .025 & .2 \\ 100 & 100 \end{bmatrix} \qquad CST_2 = \begin{bmatrix} 2 & 3 & 5 \\ .05 & .3 & .05 \\ 25 & 50 & 50 \\ 100 & 200 & 100 \end{bmatrix}$$

Step 2. (r = 0) J(2) = 2, J(1) = 1 where h = 2

$M_{12} = 1$, $MX_{12} = 4$

Step 3. (r = 0) $NCOL_2 = 3 < 4 = MX_{12}$

$MX'_{11} = 16$, $MX_{11} = 5$

$NCOL_1 = 2 < 5 = MX_{11}$

Step 4. (r = 0) $DSAV_{(1)0} = (2.5 + 20) + (6.25 + 75 + 7.5)$

$= 22.5 + 88.75 = 111.25$

Step 1. (r = 1) $CST_1 = \begin{bmatrix} 1 \\ .025 \\ 100 \end{bmatrix}$, $CST_2 = \begin{bmatrix} 2 \\ .05 \\ 25 \\ 100 \end{bmatrix}$, $CST_3 = \begin{bmatrix} 4 \\ .1 \\ 200 \\ 25 \\ 100 \end{bmatrix}$

Step 2. (r = 1) $j(3) = 3$, $j(2) = 2$, $j(1) = 1$ where $h = 3$

$M_{13} = 1$, $MX_{13} = 2$

Step 3. (r = 1) $NCOL_3 = 1 < 2 = MX_{13}$

$MX'_{12} = 8$, $MX_{12} = 3$

$NCOL_2 = 1 < 3 = MX_{12}$

$MX'_{11} = 12$, $MX_{11} = 5$

$NCOL_1 = 1 < 5 = MX_{11}$

Step 4. (r = 1) $DSAV_{(1)1} = 2.5 + 6.25 + 32.5 = 41.25$

* Step 6. (r = 1) $XDSAV_{(1)1} = 111.25 + 41.25 = 152.50$

$EXSAV_{(1)1} = ESAV_{(1)1} + XDSAV_{(1)1}$

$= 0 + 152.50 = 152.50$

* Step 5 did not need to be performed, according to the algorithm.

D. Examples and Comparisons

1. Failure of Pollack's Algorithm and Inconsistency of Verhelst's Algorithm

Consider the following table:

	P_1	P_2	P_3	P_4	P_5	PE
	.2	.2	.3	.1	.1	.1

			R1	R2	R3	R4	R5	ELSE
T_1	1	C1	-	0	0	1	1	
T_2	1	C2	0	-	0	1	0	
T_3	1	C3	0	1	-	0	1	
T_4	1	C4	0	0	1	-	-	
			A1	A2	A3	A4	A5	AE

The proposed algorithm gives the values:

Condition C1 $\quad ESAV_{(1)1} = 0 \qquad DSAV_{(1)0} = .2 + .3 = .5$

$$DSAV_{(1)1} = .1 + .1 = .2$$

$$XDSAV(1)_1 = .7$$

$$EXSAV_{(1)1} = 0 + .7 = .7$$

Condition C2 $\quad EXAV_{(2)1} = 0 \qquad DSAV_{(2)0} = .2 + .3 = .5$

$$DSAV_{(2)1} = .1$$

$$XDSAV_{(2)1} = .6$$

$$EXSAV_{(2)1} = 0 + .6 = .6$$

Condition C3 $ESAV_{(3)1}=0$ $DSAV_{(3)0}= .2 + .1 = .3$

$DSAV_{(3)1}= .2 + .1 = .3$

$XDSAV_{(3)1}= .6$

$EXSAV_{(2)1}= 0 + .6 = .6$

Condition C4 $ESAV_{(4)1}= 0$ $DSAV_{(4)0}= .2 + .2 = .4$

$DSAV_{(4)1}= .3$

$XDSAV_{(4)1}= .7$

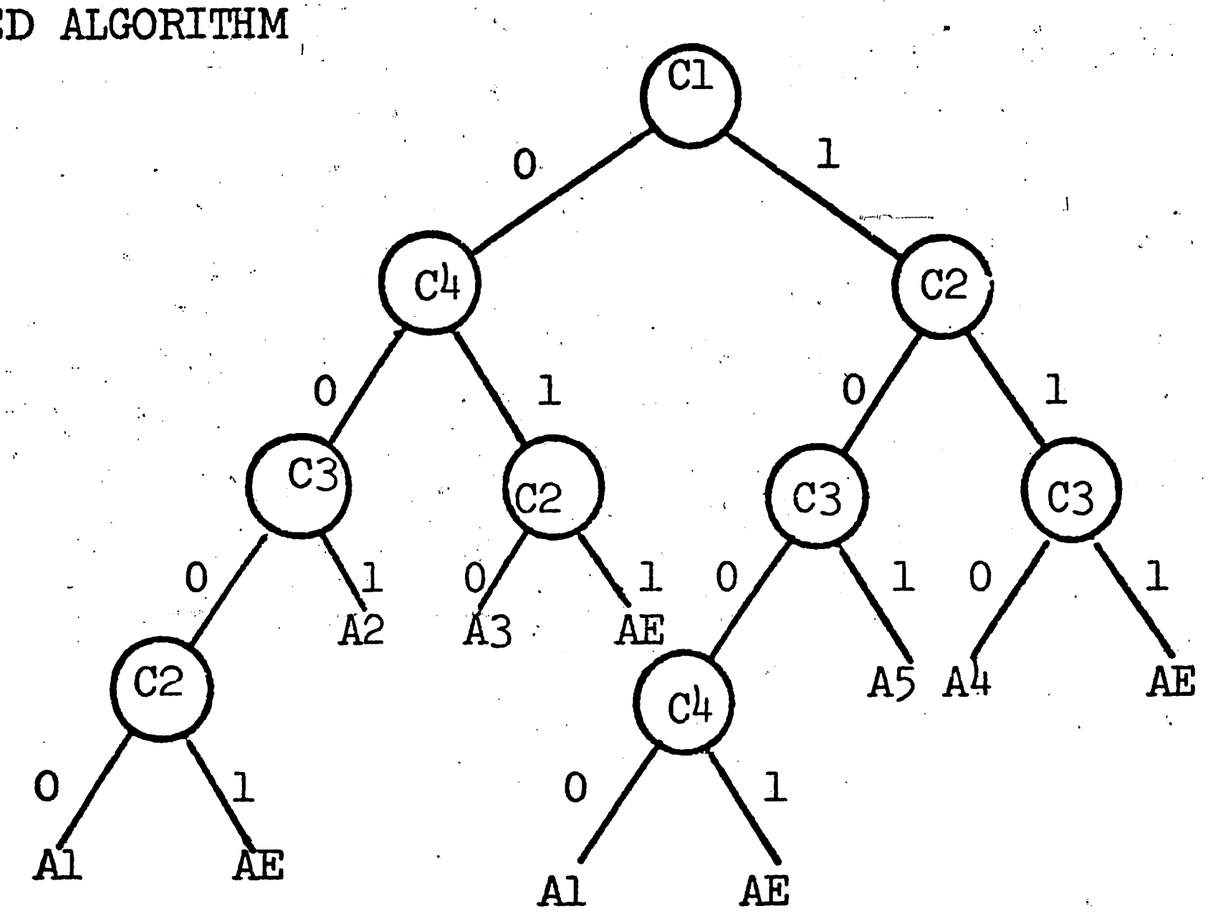
Conditions C1 and C4 are equally good choices for the root node, both offering an expected time savings of .7. Pollack's algorithm gives:

	Weighted Dash Ct.	Delta
C1	.4	.6
C2	.4	1.0 *max. delta
C3	.6	N/A
C4	.4	.2

The indication here is that C2 should be chosen for the root node. The decision trees generated by the recursive application of both algorithms are seen in Figures 4.3 and 4.4.

The algorithm by Verhelst[23] (as discussed in Chapter II), when applied to the table in Example 1, would indicate C1, C2 or C4 as equally likely choices for the root node (i.e., $T_1 = T_2 = T_4 = .2$ as in Figure 2.9, Chapter II). The recursive application of the algorithm

PROPOSED ALGORITHM



Average response time:

$$(P_1/2)(T_1 + T_4 + T_3 + T_2) + P_e(T_1 + T_4 + T_3 + T_2) + P_2(T_1 + T_4 + T_3)$$

$$+ P_3(T_1 + T_4 + T_2) + P_e(T_1 + T_4 + T_2) + (P_1/2)(T_1 + T_2 + T_3 + T_4)$$

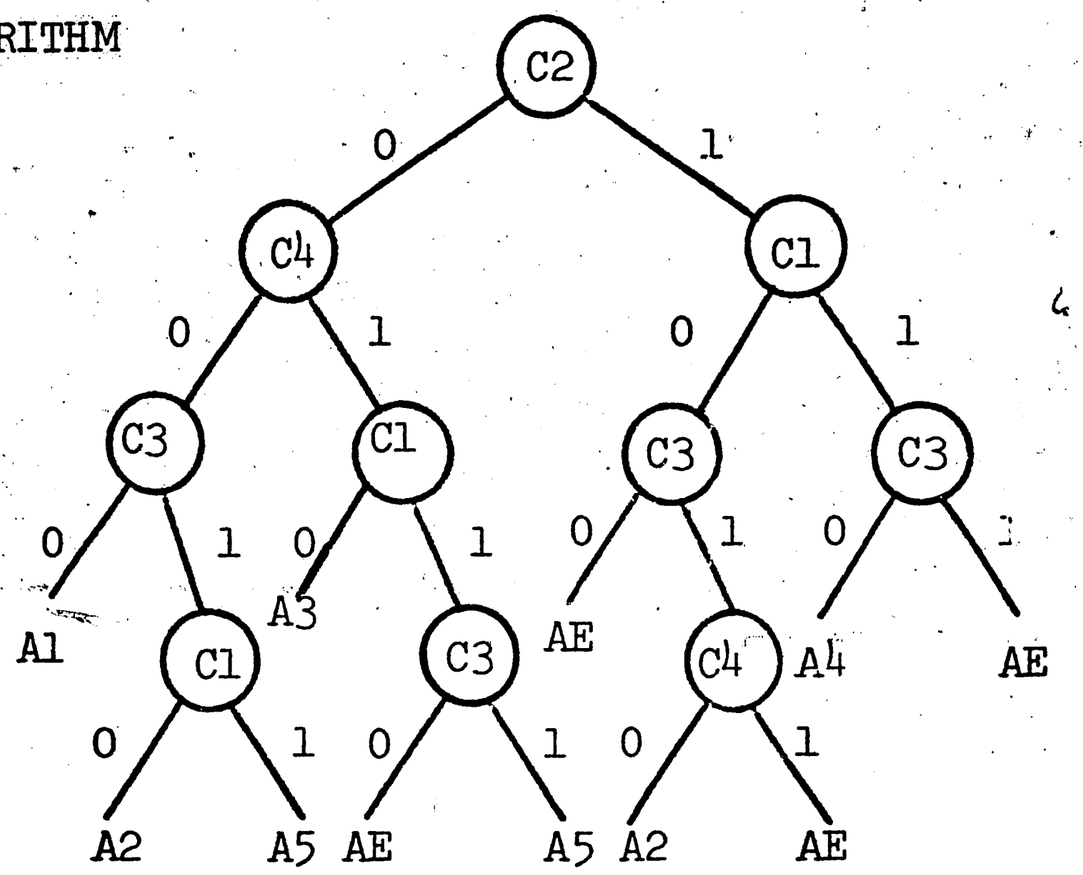
$$+ P_e(T_1 + T_2 + T_3 + T_4) + P_5(T_1 + T_2 + T_3) + P_4(T_1 + T_2 + T_3)$$

$$+ P_e(T_1 + T_2 + T_3) = .1(4) + .016(4) + .2(3) + .3(3) + .016(3) + .1(4)$$

$$+ .016(4) + .1(3) + .1(3) + .016(3) \approx \underline{3.124}$$

FIGURE 4.3

POLLACK'S ALGORITHM



Average response time:

$$P_1(T_2 + T_4 + T_3) + (P_2/2)(T_2 + T_4 + T_3 + T_1) + (P_5/2)(T_2 + T_4 + T_3 + T_1)$$

$$+ P_3(T_2 + T_4 + T_1) + P_e(T_2 + T_4 + T_1 + T_3) + (P_5/2)(T_2 + T_4 + T_1 + T_3)$$

$$+ P_e(T_2 + T_1 + T_3) + (P_2/2)(T_2 + T_1 + T_3 + T_4) + P_e(T_2 + T_1 + T_3 + T_4)$$

$$+ P_4(T_2 + T_1 + T_3) + P_e(T_2 + T_1 + T_3) = .2(4) + .1(4) + .05(4) + .3(3)$$

$$+ .016(4) + .05(4) + .016(3) + .1(4) + .016(4) + .1(3) + .016(3) \approx 3.424$$

FIGURE 4.4

to obtain trees with C1 and C2 as the root node results in the same trees as in Figures 4.3 and 4.4. These trees have different average response times. Thus, there appears to be an inconsistency in the method of determining a condition to test. However, when the proposed algorithm is recursively applied, its' alternate choice (C4) for the root node, results in a decision tree with the same average response time as the tree with C1 for the root node, This illustrates the consistency of the calculations as opposed to the Verhelst algorithm.

2. Incompleteness of Shwayder's Algorithm

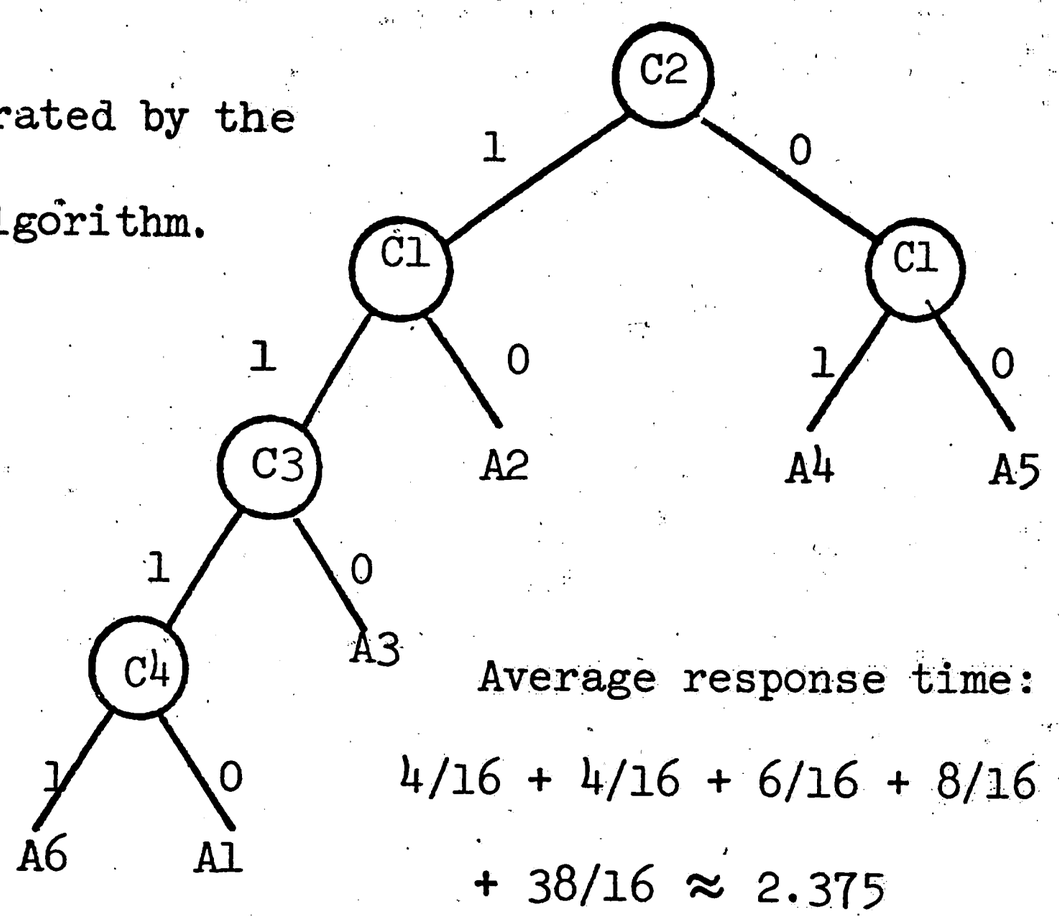
Consider the following table:

			P_1	P_2	P_3	P_4	P_5	P_6
			1/16	4/16	2/16	3/16	5/16	1/16
			R1	R2	R3	R4	R5	R6
T_1	1	C1	1	0	1	1	0	1
T_2	1	C2	1	1	1	0	0	1
T_3	1	C3	1	-	0	-	-	1
T_4	1	C4	0	-	-	-	-	1
			A1	A2	A3	A4	A5	A6

As seen in Figure 2.8, in Chapter II, the information theory algorithm by Shwayder[22] indicates C2 for the root node in a decision tree. The average response time of the tree (Figure 2.8, Chapter II) created by recursive application of Shwayder's algorithm is 2.375. Shwayder himself points out that there exists a tree with C1 as the root node

that also has an average response time of 2.375, even though his algorithm does not indicate this tree using the entropy calculations. The proposed algorithm gives the following calculations for the root node: $EXSAV_{(1)1} = 13/8$, $EXSAV_{(2)1} = 13/8$, $EXSAV_{(3)1} = 9/16$, $EXAV_{(4)1} = 9/16$. The indication here is that C1 and C2 are equally good choices for the root node. The tree created by recursive application of the proposed algorithm, having C2 as the root node, is the same as the tree in Figure 2.8. The tree having C1 as the root node is the tree refered to by Shwayder. Both have average response times of 2.375 and are shown in Figure 4.5. Therefore, the proposed algorithm indicated both trees; whereas, Shwayder's algorithm only indicated one.

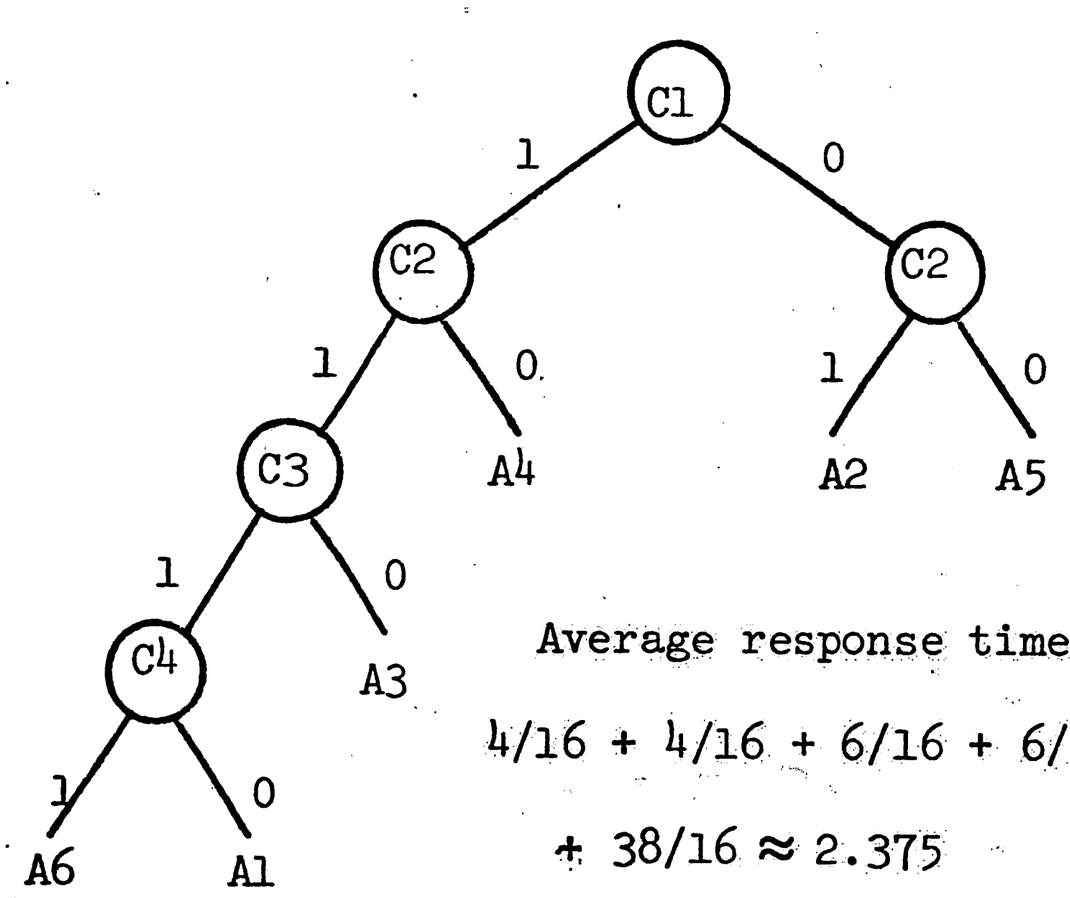
Trees generated by the proposed algorithm.



Average response time:

4/16 + 4/16 + 6/16 + 8/16 + 6/16 + 10/16

+ 38/16 ≈ 2.375



Average response time:

4/16 + 4/16 + 6/16 + 6/16 + 8/16 + 10/16

+ 38/16 ≈ 2.375

FIGURE 4.5

CHAPTER V

SUMMARY AND RECOMMENDATIONS FOR FURTHER STUDY

The term decision table optimization, as used in this thesis, pertained to the translation of a decision table into an object program under certain optimization criteria. The optimization criteria studied were:

1. Minimum Storage Considerations
2. Minimum Response (run) time Considerations.

Due to the large number of possible complete trees that must be examined when using optimal seeking methods, a heuristic approach utilizing decision trees was used. As seen in Chapter II, the existing heuristic algorithms were not preceded with any information concerning basic concepts or justification. They, for the most part, appeared to be derived from observation. Consequently, their performance varied or could be considered unstable.

When the major heuristic algorithms in the literature were compared to the proposed algorithms, one or more of the following situations existed.

1. The proposed algorithms performed better under their respective optimization criteria than existing algorithms.
2. The existing algorithms were not decisive (inconsistent) in their choices of conditions.

That is, they frequently indicated a multiple choice of conditions for a node; when, in fact, these conditions did not result in trees of the same size or response time.

3. The existing algorithms were incomplete in their choices for conditions. That is, they frequently did not indicate multiple choices.

The instability of the existing algorithms was due to several things. All pertain to the method of analysis of a decision table. As seen in Chapter III, the combination of simple rules resulted in the following:

1. Dash entries in a table
2. Provisions for an ELSE rule.

These rule combinations established the existence of an optimal partial tree. As seen in Chapter III, Part B, the ELSE rule was a major contributing factor in non-terminal node elimination. The existing algorithms ignored the ELSE rule in their calculations. The proposed algorithms had a heuristic "look-ahead" strategy designed to estimate the number of node sets that could be eliminated from a complete tree (sub-tree). For the minimum storage case, the node sets were converted to their equivalent number of non-terminal nodes. When the optimization criterion was to minimize response time, the node sets were converted to their equivalent expected test times. This was the expected time to test all the conditions in a path from the parent node of a node set to a terminal node. This provided a

measure of the test time saved when the node set is eliminated.

The existing algoritms did not ignore the dashes as they did the ELSE rule. They did, however, base their calculations on all the dashes in a table. This was done without regard to the fact that some of the dashes might be split ; thus, eliminiating the split dashes's capability for node set eliminations. The proposed algorithms tested for dash splitting. They also took in account the hierarchal effect that higher order node set eliminations had on the available number of lower order node sets. The algorithms then estimated the number of node sets that could be eliminated due to dashes in the table.

Recommendations for further study are as follows:

1. Study the effects that dash placement has on the minimum response time algorithm and develop a user oriented procedure for dash placement that will help the algorithm produce better trees.
2. Develop an algorithm that will provide tradeoffs (during the creation of a decision tree) between storage and response time such that when one criterion is used and has met its requirements (time or storage), the algorithm will then switch its optimization criterion.
3. Study the effects of the action sets and compiler efficiency on the overall problem of optimization.
4. Investigate the possibility of extending the concepts developed in this thesis to extended entry tables.

BIBLIOGRAPHY

1. Abramson, N., Information Theory and Coding, McGraw Hill, N.Y., 1963.

2. Ellis, J., Western Electric Decision Table Manual, A Users Guide, June 15, 1968 (Revised).

3. Fano, Robert M., "The Transmission of Information", MIT Research Lab of Electronics, Tech Report No. 65, March 1949.

4. Ganapathy, S. and Rajaraman, V., "Information Theory Applied to the Conversion of Decision Tables to Computer Programs", Unpublished paper.

5. Garey, M. R., "Optimal Binary Identification Procedures", SIAM Journal of Applied Mathematics, Vol. 23, No. 2, September, 1972.

6. King, P. J., "Conversion of Decision Tables to Computer Programs by Rule Mask Techniques", Comm. of the ACM, Nov., 1966.

7. King, P. J., "Decision Tables", The Computer Journal, Vol. 10, No. 2, Aug., 1967.

8. Kirk, H. W., "Use of Decision Tables in Computer Programming", Comm. of the ACM, Jan., 1965.

9. McDaniel, H., Decision Table Software, Brandon Sys. Press Inc.

10. Montalbano, M., "Tables, Flow Charts and Program Logic", IBM Systems Journal, Sept., 1962.

11. Muthukrishnan, C. R., "On the Conversion of Decision Tables to Computer Programs", Comm. of the ACM, June, 1970.

12. Myers, H. J., "Compiling Optimized Code From Decision Tables", IBM Systems Journal, Sept., 1972.

13. Peel, R., "Decision Table Translators", The Computer Bulletin, Dec., 1969.

14. Pollack, S. L., "Conversion of Limited Entry Decision Tables to Computer Programs", Comm. of the ACM, Nov., 1965.

15. Pollack, S. L., Decision Tables: Theory and Practice, Wiley Interscience New York, 1971

16. Pollack, S. L., "Conversion of Limited Entry Decision Tables to Computer Programs", Memorandum RM-4020-PR, U. S. Air Force Project Rand, May, 1964.

17. Pollack, S. L., "Analysis of the Decision Rules in Decision Tables", U. S. Air Force Project Rand, May, 1963.

18. Press, L. I., "Conversion of Decision Tables to Computer Programs", Comm of the ACM, June 1965.

19. Rabin, J., "Conversion of Limited Entry Decision Tables Into Optimal Decision Trees: Fundamental Concepts", Sigplan Notices, Sept., 1971.

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

21. Reinwald, L. T. and Soland, R. M., "Part II, Minimum Storage Requirements", Journal of the ACM, Oct., 1967.

22. Shwayder, K., "Conversion of Limited Entry Decision Tables to Computer Programs", Comm. of the ACM, Feb., 1971.

23. Verhelst, M., "The Conversion of Limited Entry Decision Tables to Optimal and Near-Optimal Flowcharts: Two New Algorithms", Comm. of the ACM, Nov., 1972

24. Yasu, Tashio, "Some Aspects of Decision Table Conversion Techniques", Sigplan Notices, Sept., 1971.

VITA

PERSONAL HISTORY

NAME: Wallace Lee Dingee Jr.

DATE OF BIRTH: January 7, 1948

PLACE OF BIRTH: Baltimore Maryland

PARENTS: Lee and Rosemary Dingee

WIFE: Joy R. Dingee

EDUCATIONAL BACKGROUND

C. E. Byrd High School
Shreveport, Lousiana -- Graduated 1965

Louisiana Tech University
Bachelor of Science in Mathematics
Graduated 1969

Lehigh University
Candidate for Master of Science in Industrial Engineering
1971 - 1973

HONORS

Alpha Pi Mu

PROFESSIONAL EXPERIENCE

Western Geophysical Corporation
Shreveport, Louisiana
Systems Programmer -- Summer 1967

Western Electric Corporation
Shreveport, Louisiana Works
Special Tech. Assistant -- Summer 1968

Western Electric Corporation
Shreveport, Louisiana Works
Information Systems Staff Member - 1969 - 1971

Lehigh Master's Program
Western Electric Corporate
Education Center - 1971 - 1973