ret to Borod

~~[illegible]~~

~~[illegible]~~

~~ret to Borod~~

9/17

File Dec tables

MEMORANDUM
RM-4020-PR
MAY 1964

CONVERSION OF LIMITED-ENTRY DECISION TABLES TO COMPUTER PROGRAMS

Solomon L. Pollack

PREPARED FOR:
UNITED STATES AIR FORCE PROJECT RAND

The RAND Corporation
SANTA MONICA • CALIFORNIA

MEMORANDUM

RM-4020-PR

MAY 1964

CONVERSION OF LIMITED-ENTRY DECISION TABLES TO COMPUTER PROGRAMS

Solomon L. Pollack

This research is sponsored by the United States Air Force under Project RAND—contract No. AF 49(638)-700 monitored by the Directorate of Development Planning, Deputy Chief of Staff, Research and Development, Hq USAF. Views or conclusions contained in this Memorandum should not be interpreted as representing the official opinion or policy of the United States Air Force.

The RAND Corporation
1700 MAIN ST. • SANTA MONICA • CALIFORNIA • 90406

PREFACE

This Memorandum is the fourth in a series of RAND publications on decision tables for the Air Force. S. L. Pollack and K. R. Wright, Data Description for DETAB-X (Decision Table, Experimental), RM-3010-PR, March 1962, described the specifications for data description for DETAB-X, an experimental decision table computer language. S. L. Pollack, Analysis of Decision Rules in Decision Tables, RM-3669-PR, May 1963, developed a theory for decision tables and described how to analyze them for completeness, redundancy and contradiction of decision rules. S. L. Pollack, How to Build and Analyze Decision Tables, P-2829, November 1963, described the initial steps one must take to develop decision tables, a method for combining pairs of decision rules, and again how to analyze them for completeness, redundancies and contradictions.

Like the earlier publications, this Memorandum is one more step toward the goal of developing the decision tables as a useful tool for Air Force management. But unlike the others, it is intended primarily for Air Force data processing specialists -- system analysts and computer programmers.

SUMMARY

Decision tables are useful for describing a set of complex decision rules based on given sets of conditions. Algorithms that can efficiently convert the tables into computer programs will extend the usefulness of decision tables to computer users. This Memorandum describes two such algorithms, based on work done by M. S. Montalbano and extended here to handle dashes and ELSE-decision rules. The first algorithm minimizes the computer storage space required for the resultant program, the second minimizes computer running time. During the conversion process, both pinpoint any contradictions or redundancies among the rules in a table.

A necessary adjunct to minimizing computer storage or running time is the allowable reduction of the number of rules in a decision table. This Memorandum describes a technique to effect this reduction for pairs, triplets and quadruplets of rules. The system analyst will find this method most helpful for pairs, and generally unprofitable for n-tuplets greater than three. The technique can be done manually or accomplished by the computer as a prelude to executing one of the two algorithms.

CONTENTS

I. INTRODUCTION

The usefulness of decision tables for writing computer source programs can be increased by the availability of algorithms that can convert any limited-entry decision table* into a sequence of individual comparisons and the series of actions associated with both branches of each comparison. This paper describes two such algorithms. One minimizes the number of comparison instructions required in the program, thus reducing the computer storage space required. The other minimizes the number of comparison instruction executions, thus reducing the total computer running time.

The availability of these algorithms gives the computer programmer a choice between the two for each of his decision tables. For example, the programmer might specify the first algorithm for those tables executed only a few times, thus possibly sacrificing some computer running time to reduce the amount of memory storage required for his program. For those tables used in a tight loop, i.e., executed many times, he will probably use the second algorithm to cut down computer running time at the possible sacrifice of using additional computer storage.

Before converting the decision table to individual comparisons and the series of actions associated with each branch, the number of written decision rules that result in the same series of actions must be reduced to a minimum. As an example,**

	R_1	R_2	R_3	R_4
C_1	Y	Y	N	N
C_2	N	Y	Y	N
C_3	Y	N	Y	Y
A_1	x	–	–	x
A_2	–	x	x	–

*For a description of decision table structure, see S. L. Pollack, Analysis of Decision Rules in Decision Tables, The RAND Corporation, RM-3669-PR, May 1963.

**For consolidation of two rules with identical actions, see S. L. Pollack, How to Build and Analyze Decision Tables, The RAND Corporation, P-2829, November 1963.

is equivalent to

	$R_{1,4}$	R_2	R_3
C_1	-	Y	N
C_2	N	Y	Y
C_3	Y	N	Y
A_1	x	-	-
A_2	-	x	x

This latter table would then be converted to an object program by the algorithms described later in this paper.

Section II discusses three procedures for reducing the number of written rules in a decision table. And Section III explains the algorithms for converting the decision tables to computer programs.

II. PROCEDURES FOR REDUCING THE NUMBER OF WRITTEN DECISION RULES IN A DECISION TABLE

After a system analyst has described the individual decision rules, he can reduce the number of written rules by grouping together those that specify the same set of actions, and applying to each group the following procedures.

First, any redundant rules must be eliminated. This is done by locating those pairs that do not have at least one Y, N pair in one of their rows.* In such cases (A), the two rules are decomposed (B); the redundant rule(s) eliminated (C), i.e. 2b duplicates 1b; and the remaining rules compressed, if possible (D).

(A)

RULE	
1	2
Y	Y
N	-
-	N
Y	Y
N	-

(B)

RULE					
1A	1B	2A	2B	2C	2D
Y	Y	Y	Y	Y	Y
N	N	Y	N	Y	N
Y	N	N	N	N	N
Y	Y	Y	Y	Y	Y
N	N	Y	N	N	Y

(C)

RULE				
1A	1B	2A	2C	2D
Y	Y	Y	Y	Y
N	N	Y	Y	N
Y	N	N	N	N
Y	Y	Y	Y	Y
N	N	Y	N	Y

(D)

RULE		
1	2A,2C	2D
Y	Y	Y
N	Y	N
-	N	N
Y	Y	Y
N	-	Y

To arrive at step D, the system analyst checks all pairs of rules in the group. A pair of rules is equivalent to (can be combined into) one rule if the pair agrees in every row but one, and if, in that row, one rule contains a Y and the other contains an N. Then the rule is set up like the following example, with the same entry as the original pair in corresponding rows, and a dash in the dissimilar row.

(A) $$\begin{bmatrix}Y\\Y\\Y\\N\end{bmatrix},\begin{bmatrix}Y\\Y\\N\\N\end{bmatrix}\longrightarrow\begin{bmatrix}Y\\Y\\-\\N\end{bmatrix}$$

(B) $$\begin{bmatrix}-\\N\\Y\\-\end{bmatrix},\begin{bmatrix}-\\Y\\Y\\-\end{bmatrix}\longrightarrow\begin{bmatrix}-\\-\\Y\\-\end{bmatrix}$$

*Pollack, Analysis of Decision Rules in Decision Tables.

For those interested in further reducing the number of written rules, the following procedure is offered for combining into pairs, triplets of rules that specify the same series of actions. In many cases, reducing three rules to two will not be worth the additional effort involved. However, the algorithms for converting decision tables to optimal object program do require that the table contain the minimum number of written rules.

The procedure for converting triplets to pairs first locates pairs, one of which can be split into two rules, by virtue of a dash row. Two cases are then possible.

(1) One of the rules resulting from the split can combine with the second rule of the original pair. The other rule of the split can be combined with a third rule, if available.

(2) One of the rules resulting from the split can combine with the second rule of the original pair, and this resultant rule can be combined with a third rule, if available.

Case 1 can be achieved by executing the following procedure. With each rule that has one or more dashes, test every other rule in the table to locate a second rule that agrees with it in every row but one (excluding the dash row). In that row of disagreement one rule must contain a Y, the other an N. In the dash row, the second rule must contain either Y or N, but no dash.

Then search among the remaining rules for a third rule that disagrees with the second rule in the dash row, one having a Y, the other an N. Compare the first and third rules. If they agree in every row but one (excluding the dash row), and if, in that row, there is a Y, N pair, then the three rules are equivalent and can be combined into two rules (A in the following example). Split the first rule into two (B), and combine the appropriate pairs (C).

(A) (B) (C)

$$\left[\begin{array}{c|c|c} R_1 & R_2 & R_3 \\ \hline Y & Y & Y \\ Y & Y & N \\ - & N & Y \\ N & Y & N \end{array}\right] \rightarrow \left[\begin{array}{c|c|c|c} R_1 & R_1 & R_2 & R_3 \\ \hline Y & Y & Y & Y \\ Y & Y & Y & N \\ Y & N & N & Y \\ N & N & Y & N \end{array}\right] \rightarrow \left[\begin{array}{c|c} R_{1,3} & R_{1,2} \\ \hline Y & Y \\ - & Y \\ Y & N \\ N & - \end{array}\right]$$

Case 2 can be achieved by executing the following procedure. Against each rule that has one or more dashes, test every other rule in the table to locate a second rule that agrees with it in every row but one (excluding the dash row). In that row of disagreement one must contain a Y, the other an N. The second rule must contain either a Y or N in the dash row.

Then search among the remaining rules for a third rule that agrees with the second rule in the dash row, that contains a dash in the row where the Y, N pair appeared for the first and second rules, and that contains a Y, N pair in conjunction with the first rule, agreeing everywhere else (A in the following example). Split the first rule into two rules (B). Combine one of them with the second rule (C), and combine the resultant rule with the third rule (D).

(A) (B) (C) (D)

$$\left[\begin{array}{c|c|c} R_1 & R_2 & R_3 \\ \hline Y & Y & N \\ N & Y & - \\ - & Y & Y \end{array}\right] \rightarrow \left[\begin{array}{c|c|c|c} R_1 & R_1 & R_2 & R_3 \\ \hline Y & Y & Y & N \\ N & N & Y & - \\ Y & N & Y & Y \end{array}\right] \rightarrow \left[\begin{array}{c|c|c} R_{1,2} & R_1 & R_3 \\ \hline Y & Y & N \\ - & N & - \\ Y & N & Y \end{array}\right] \rightarrow \left[\begin{array}{c|c} R_{1,2,3} & R_1 \\ \hline - & Y \\ - & N \\ Y & N \end{array}\right]$$

Note that in reducing triplets to pairs, a pair was required that met all the requirements for combining two into one, except that in one of the rows that should have had total agreement, one rule had a dash, the other a Y or N. In reducing quadruplets to triplets, again it is necessary to meet all the requirements for converting triplets to pairs, except that one of the triplets will have a dash instead of the Y or N that should be there. This rule can be split in two and one half joined to a fourth rule. This searching can be done for any n-tuplet. Generally, however, the system analyst will not find it profitable to attempt to reduce n-tuplets greater than three. In most cases the analyst will be satisfied to reduce pairs of rules to a single rule.

III. ALGORITHMS FOR CONVERTING LIMITED-ENTRY DECISION TABLES TO COMPUTER PROGRAMS

To economize in writing this section, the author omitted the actions in the decision tables and in the flow charts that follow. Each rule number (R_1, R_2, R_3,...) shown in the flow charts can be replaced by the series of actions specified for each decision rule. In addition, any rules not specified or implied are assumed to be part of an ELSE-Rule not shown in the table.

To help the reader follow the two algorithms described below, we first present a general description of how to convert the decision table to flow chart form.

One row of the original decision table is selected -- the criterion for selection differs for the two algorithms.* The condition in that row becomes the first comparison of the flow chart. The original decision table is then decomposed into two subtables (containing one less row), or one subtable and one rule, or two rules, and each of these is associated with each branch of the comparison.

A row is then selected from each of these subtables and its condition becomes attached to the previously selected condition, as will be explained later. This process is continued until each rule of the original decision table or an ELSE-Rule appears as one of the branches of a condition, or a subtable indicates that the original table contained redundant or contradictory rules.

We now describe the 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 algorithms are based upon "the quick rule" and "delayed rule" methods described in M. S. Montalbano's "Tables, Flow Charts, and Program Logic," IBM Systems Journal, September 1962. This Memorandum extends these to handle dashes and ELSE-decision rules.

ALGORITHM 1

Objective

To convert a decision table to a computer program and have this program use the minimum number of storage locations.

Steps in the Algorithm

Step 1. Check for redundancy or contradiction. If at any stage, a pair of rules does not contain at least one Y, N pair in any of its rows, redundancy or contradiction exists. Such is the case for Rules 1 and 2 below.

	R_1	R_2
C_1	-	N
C_2	N	N

Depending on whether or not these rules have identical actions, they either contradict each other or contain redundant rule(s). If the actions for Rules 1 and 2 are the same, Rule 2 is redundant. If the actions for Rules 1 and 2 are different, the rules contradict each other.

Rather than check for redundancy or contradiction at every stage, it is better to wait until a table is reduced to one condition.

	R_1	R_2	R_3
C_5	-	Y	N

If there are more than two rules for a table with only one condition, contradiction or redundancy exists. Where the actions for Rule 1 differ from those of Rules 2 and 3, then either 2 or 3 is contradictory. Where the actions of Rule 1 are the same as those of Rules 2 or 3, then either 2 or 3 is redundant.

Step 2. Determine those rows that have a minimum dash-count. A dash that appears in a rule (i.e. column) that contains r dashes is counted as 2^r dashes. For each rule, we denote the 2^r as the column-count. In each row, the sum of the column-counts corresponding to the dashes in that row is denoted as dash-count. A row's dash-count is the sum of the column counts of those rules that have dash entries in the row. This is illustrated in the following example, in which Row 2 has the minimum dash-count.

COLUMN-COUNT =	4	4	2	
	R_1	R_2	R_3	DASH-COUNT
C_1	N	-	Y	4
C_2	N	Y	-	2
C_3	-	Y	N	4
C_4	-	-	N	8

Step 3. If two or more rows have a minimum dash-count, select that row that has the maximum delta, which is the absolute value of the difference of the Y-count and the N-count. The Y-count is the sum of the column-counts corresponding to the Y's in the row. The N-count is similar.

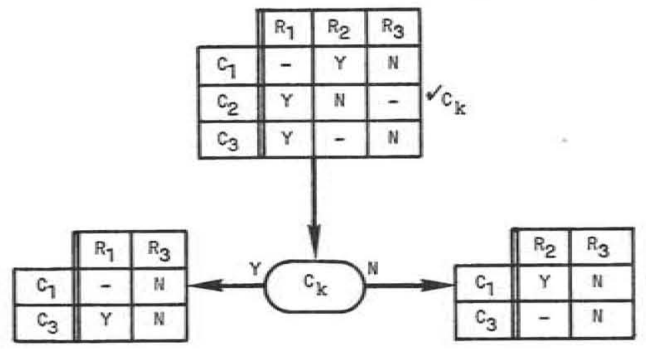
COLUMN-COUNT =	1	2	0	2	2	2		
	R_1	R_2	R_3	R_4	R_5	R_6	DASH-COUNT	DELTA
C_1	Y	N	N	N	N	-	2	5
C_2	Y	N	Y	-	Y	N	2	0
C_3	N	N	N	Y	-	Y	2	0
C_4	N	-	N	N	Y	Y	2	0

All rows in the example have the minimum dash-count, so we check their deltas. Since C_1 has the maximum delta, it is selected. The selected row is called k. In the following example $C_2 = C_k$.

COLUMN-COUNT =	2	1	1			
	R_1	R_2	R_3	DASH-COUNT	DELTA	
C_1	-	Y	N	2		
C_2	Y	Y	Y	0	4	C_k
C_3	Y	N	N	0	0	

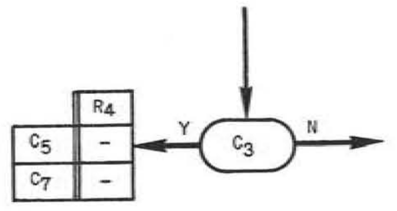
Step 4. Discriminate on the condition in Row k; call it C_k. This discrimination has two branches, each of which leads to a sub-table containing one or more rules, with one row less than the original table (Row k is deleted).

As illustrated below, the Y-Branch leads to a subtable containing all rules from the previous table that had a Y in row k. The N-branch works in a similar fashion. In addition, both subtables contain those rules that had a dash in row k of the previous table.

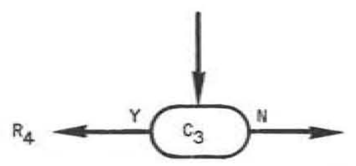


Step 5. If a branch leads to a subtable containing more than one rule, go back to Step 1.

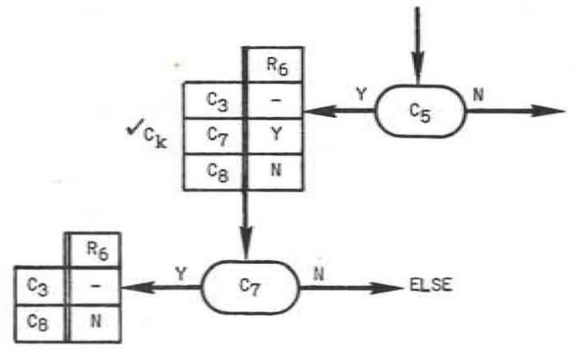
Step 6a. If a branch leads to a subtable containing exactly one rule, and if that rule contains all dashes, then replace the subtable with the rule itself.



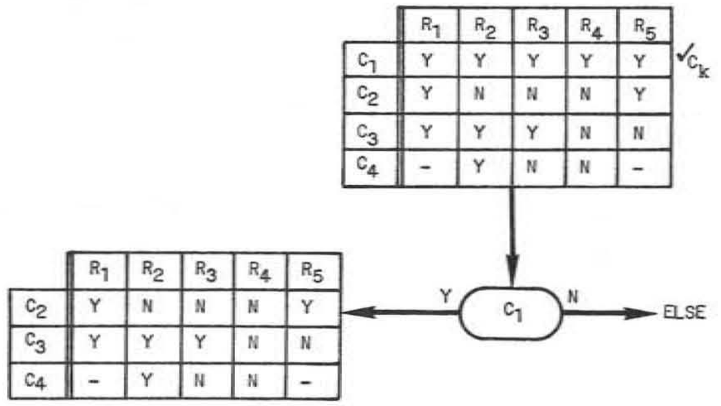
This becomes:



Step 6b. If a branch leads to a subtable with one rule, and that rule contains one or more written dashes but does not contain all dashes, choose as Row k any row that has no dash. The selected branch will indicate a subtable with one less row in it. The opposing branch of the selected row will indicate an ELSE-Rule.



Step 6c. If a branch doesn't lead to a subtable, it leads to an ELSE-Rule.

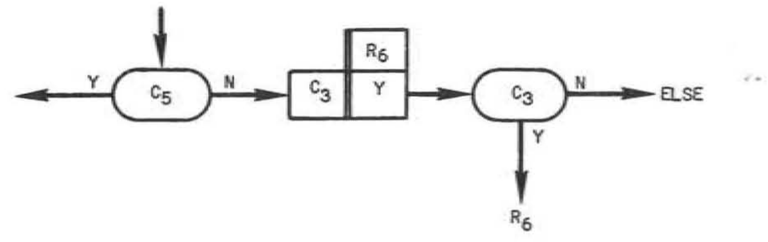


	R_1	R_2	R_3	R_4	R_5	
C_1	Y	Y	Y	Y	Y	✓C_k
C_2	Y	N	N	N	Y	
C_3	Y	Y	Y	N	N	
C_4	-	Y	N	N	-	

	R_1	R_2	R_3	R_4	R_5
C_2	Y	N	N	N	Y
C_3	Y	Y	Y	N	N
C_4	-	Y	N	N	-

Step 6d. If a branch leads to a subtable containing only one rule, and that rule contains only one condition whose value is Y (or N), then one branch of the discrimination on that condition leads to the rule,

the other branch leads to the ELSE-Rule.



We illustrate the procedure for Algorithm 1 in Fig. 1. Rules not specified in the written table are assumed to be contained in an ELSE-Rule.

ALGORITHM 2

Objective

To convert a decision table to a computer program whose comparisons can be executed in minimum time.

Assumptions

(1) Any rules not specified or implied in the table are assumed to be part of an ELSE-RULE (not necessarily shown in the table).

(2) System analysts can provide estimates of how often each rule in the table will be satisfied by an average batch of transactions to be tested.

(3) Relatively few transactions will satisfy the ELSE-Rule.

Definitions

(1) Again, column-count equals 2^r where r is the number of dashes in a column.

(2) Estimated relative frequency equals f.

(3) Weighted dash-count (WDC) for any row is the sum of the products of f and the column-counts for those columns that contain a dash in that row.

(4) Dash-count for any row is the sum of the column-counts for that row.

(5) Again, delta is the absolute value of the difference of the Y-count and the N-count.

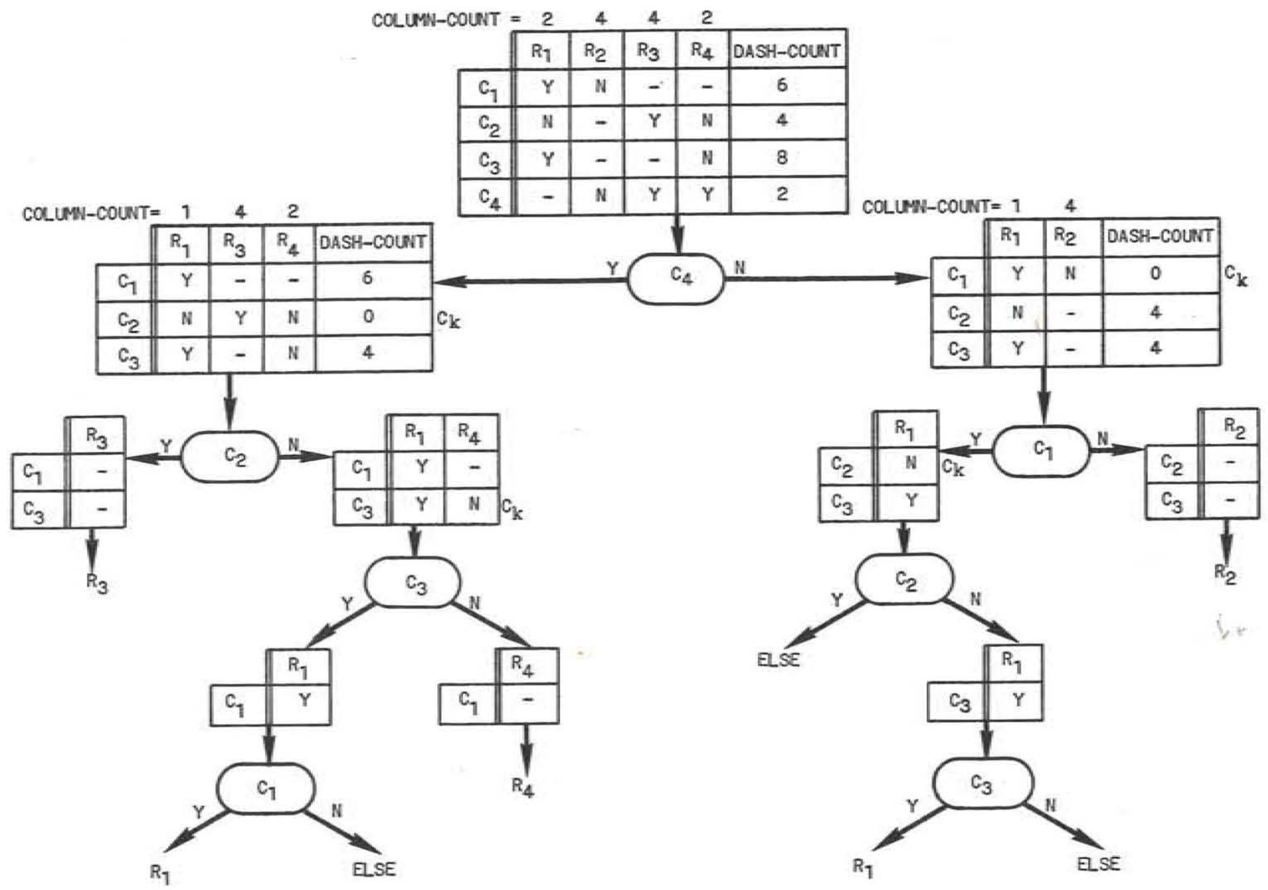
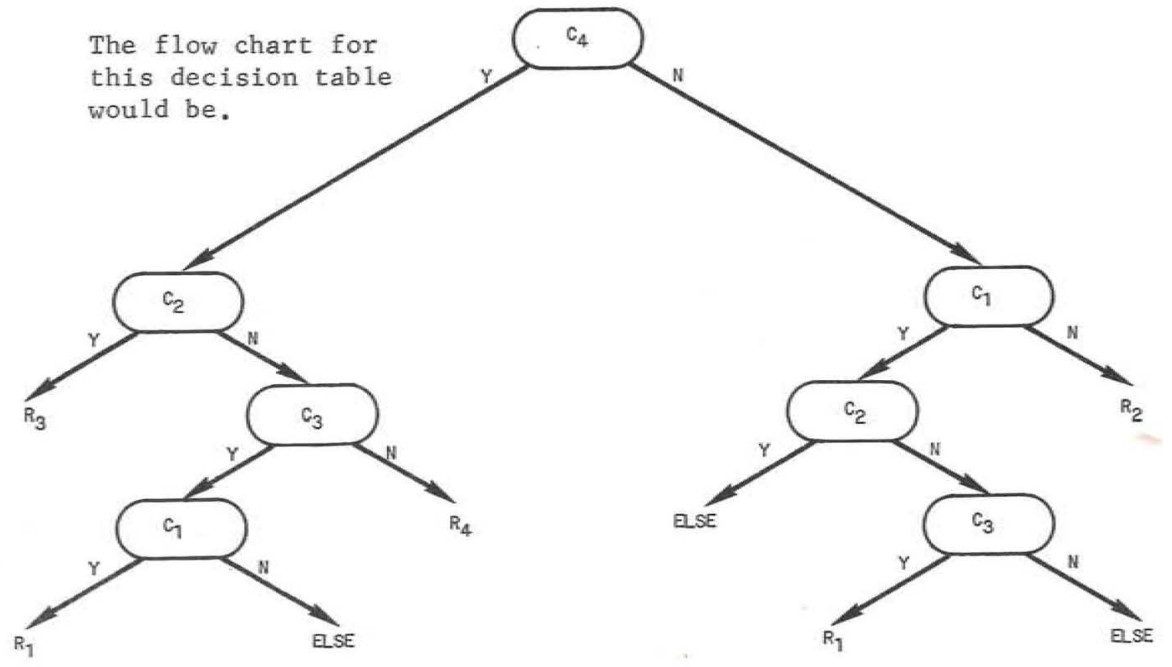


Fig. 1 -- Transformation of a Decision Table to a Series of Comparisons by Algorithm 1.

The flow chart for this decision table would be.



The above definitions are illustrated in the following example.

	R_1	R_2	R_3	R_4	WDC	DELTA	DASH-COUNT
COLUMN-COUNT =	2	2	2	1			
f =	50	30	15	5			
C_1	-	Y	-	N	130	1	4
C_2	Y	N	Y	N	0	1	0
C_3	N	-	Y	Y	60	1	2

Steps in the Algorithm

Step 1. Same as for Algorithm 1.

Step 2. Determine those rows that have a minimum weighted dash-count.

Step 3. If two or more rows have a minimum WDC, select from among them the row that has the minimum delta. If among these there still exists two or more rows, select the row with the minimum dash-count. If there are more than two such rows, select any one of them. The test on dash-count does not affect computer running time, but can save memory space without adding to computer running time.

	R_1	R_2	R_3	R_4	WDC	DELTA	DASH-COUNT
COLUMN-COUNT =	4	4	4	4			
f =	40	20	20	20			
C_1	-	Y	N	Y	160	4	
C_2	Y	-	N	-	160	0	8
C_3	Y	N	-	-	160	0	8
C_4	Y	-	-	N	160	0	8
C_5	-	Y	Y	N	160	4	

Since all rows in the example have the minimum weighted dash-count, test their deltas. Since C_2, C_3, and C_4 have the minimum delta, test their dash-counts. Since all three eligible rows (2,3, and 4) have the minimum dash-count, select any one of them as C_k.

Steps 4, 5 and 6. These steps are the same as for Algorithm 1.

We illustrate the procedure for Algorithm 2 in Fig. 2.

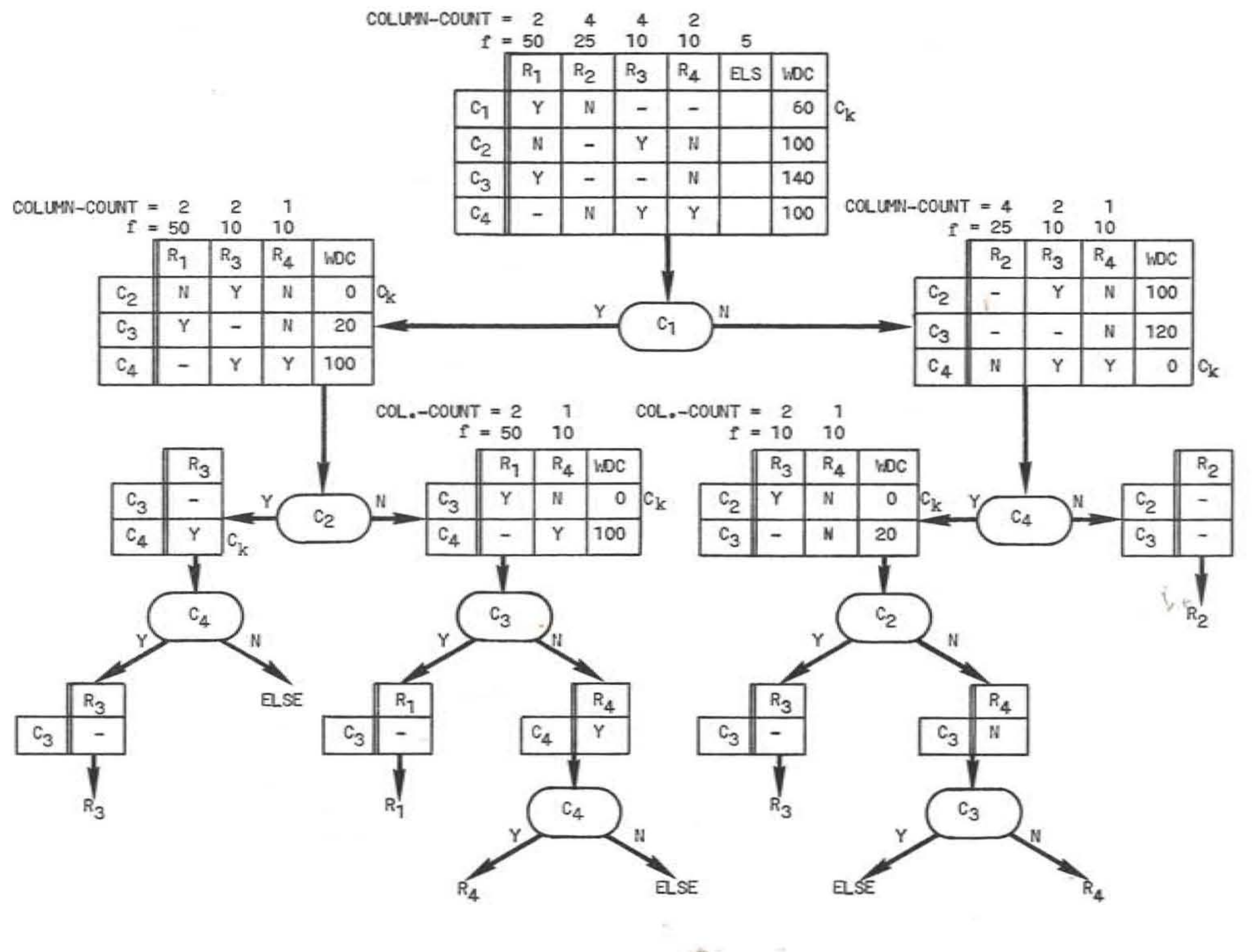
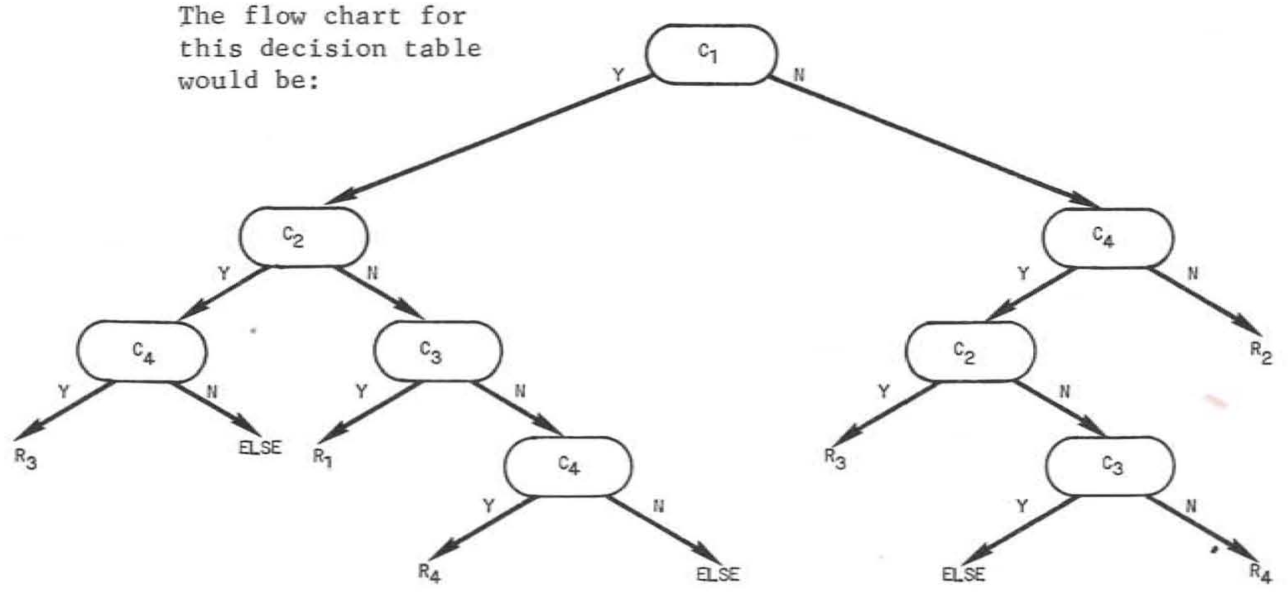


Fig. 2 -- Transformation of a Decision Table to a Series of Comparisons by Algorithm 2.

The flow chart for this decision table would be:



Note: For the transformation shown in Fig. 1, a batch of 100 transactions that satisfy Rules 1-4 in the proportion 50, 25, 10, 10, respectively (and 5 for ELSE) will require a minimum 318 comparison executions, for the transformation shown in Fig. 2, the transactions will require a minimum 288 comparison executions. See Table 1.

Table 1

COMPARISON EXECUTIONS OF FIGS. 1 AND 2

Rule No.	Fig. 1		Fig. 2	
	a	b	a	b
1	4 x	50 = 200	3 x	50 = 150
2	2 x	25 = 50	2 x	25 = 50
3	2 x	10 = 20	3 x	10 = 30
4	3 x	10 = 30	4 x	10 = 40
ELSE	(3.6)[c]x	5 = 18	(3.6)x	5 = 18
Total		318		288

a. number of comparisons
b. expected frequency
c. average number of comparisons for 3 ELSE branches