DESIGNING EXTENDED ENTRY DECISION TABLES AND OPTIMAL DECISION TREES USING DECISION DIAGRAMS

by

R. S. Michalski

Report No. 898, Department of Computer Science, University of Illinois, Urbana,
March 1978.

Report No. UIUCDCS-R-78-898 UILU-ENG 78 1708

DESIGNING EXTENDED ENTRY DECISION TABLES AND OPTIMAL DECISION TREES USING DECISION DIAGRAMS

by

Ryszard S. Michalski

March 1978

Report No. UIUCDCS-R-78-898

DESIGNING EXTENDED ENTRY DECISION TABLES AND OPTIMAL DECISION TREES USING DECISION DIAGRAMS

by

Ryszard S. Michalski

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

March 1978

This work was supported in part by the National Science Foundation under grant NSF MCS 76-22940.

ABSTRACT

The paper introduces the concept of a decision diagram and shows its application to designing extended entry decision tables and converting them to space or time optimal decision trees. A decision diagram is a geometrical representation of a decision table by means of a planar model of a multidimensional discrete space as described in [12].

Two algorithms for optimal (or suboptimal) space or time conversion are described using decision diagrams. These algorithms are basically decomposition algorithms, but by varying their degree (def. 5), one can obtain a spectrum of algorithms, differing in the trade-off between the computational efficiency and the degree of guarantee that the solution is optimal. When the algorithms do not guarantee the optimality, they give a measure of the maximum possible distance between the obtained and the optimal trees.

Key words and phrases: Limited Entry Decision Tables, Extended Entry Decision Tables, Decision Trees, Conversion Algorithms, Decision Diagram, Logic Diagram.

CR Categories: 8.3

I. INTRODUCTION

There are many practical problems where certain actions or decisions depend on the outcomes of a set of tests. A convenient way of specifying the correspondence between test outcomes and the actions is by means of a decision table. Decision tables have found a widespread application in computer programming [7,5], data documentation [3], and in various other areas of data processing. Recently, in a modified form, they have also found an application to certain problems in artificial intelligence [13]. Fig. 1 gives an example of a limited entry decision table, where tests can have only three possible outcomes: YES, NO or IRRELEVANT, denoted in Fig. 1, by 1, 0, -, respectively. Fig. 2 gives an example of an extended entry decision table, where tests can have an arbitrary number of outcomes. Techniques described in this paper are applicable to both, limited and extended entry decision tables.

Each column of a decision table specifies a decision rule which consists of a condition part (a combination of test outcomes) and an action part (an action or sequence of actions which should be taken when the condition part is satisfied). If the order of actions is important, the entries in the action part are integers indicating the order.

In any decision table, test outcomes can take only a finite number of distinct values. Let $x_1, x_2, \ldots, x_n$ denote tests and $D_1, D_2, \ldots, D_n$, corresponding sets of possible outcomes of these tests. The event space

$$\mathbf{E} = D_1 \times D_2 \times \ldots \times D_n \qquad (1)$$

(where x denotes cartesian product), is the set of all possible sequences of test outcomes (events).

As was described in [12] the event space **E** can be represented geometrically on a plane in the form of a diagram. For the lack of space, the description of the diagram, and of the rule for recognizing cartesian complexes (see below) in it, also given in [12] is omitted here.

		Rules																								
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Tests	x_1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1	1	1	1
	x_2	0	0	0	1	1	1	0	1	1	0	1	1	1	-	0	1	1	1	1	0	0	0	1	1	0	1	1
	x_3	-	1	0	-	0	1	-	-	-	-	0	-	1	-	-	-	0	-	1	0	-	-	-	0	-	-	1
	x_4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
	x_5	0	0	0	0	0	0	1	1	1	0	0	0	0	1	-	0	1	1	1	1	1	1	1	1	0	0	1
	x_6	0	-	1	0	1	1	-	0	1	-	-	1	0	-	-	-	0	1	-	-	0	1	0	1	-	-	1
Actions	A_1	1	1	1	1	1	1																					
	A_2				1	1	1	1	1	1																		
	A_3										1	1	1	1	1													
	A_4															1	1	1	1									
	A_5																			1	1	1	1	1	1	1	1	1

A limited entry decision table.

Figure 1.

		Rules													
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Tests	x_1	-	0	0	0	1	2	1	0	1	0	2	2	2	1	
	x_2	0	2	1	2	1	1	2	2	2	2	2	2	2	2	Else
	x_3	0	2	-	3	0	-	1	0	3	1	0	2	2	2	
	x_4	2	0	-	0	0	-	-	0	-	0	-	0	2	1	
Actions	A_1	1	1		1				1							
	A_2			1		1	1					1	1	1		
	A_3							1								*Error

*Error-an impossible combination of test outcomes

An extended entry decision table.

Figure 2.

It is therefore recommended that the reader have a prior acquaintance with paper [12].

A basic concept used here is that of an elementary cartesian complex (a special case of a 'cartesian complex'[12]), defined as a set of events or cells * of a diagram, which can be expressed as a single logical product (a term) of conditions which check whether a test x_i has outcome a_i. Such conditions are written as $[x_i = a_i]$, and terms as products $\Lambda_{i\in I}[x_i = a_i]$. If an outcome of a test is irrelevant ('-'), then the condition involving this test is omitted from the term.

Thus, each condition part of a rule in a decision table can be expressed as a term, and be represented in a diagram as an elementary cartesian complex (from now on, simply, a complex).

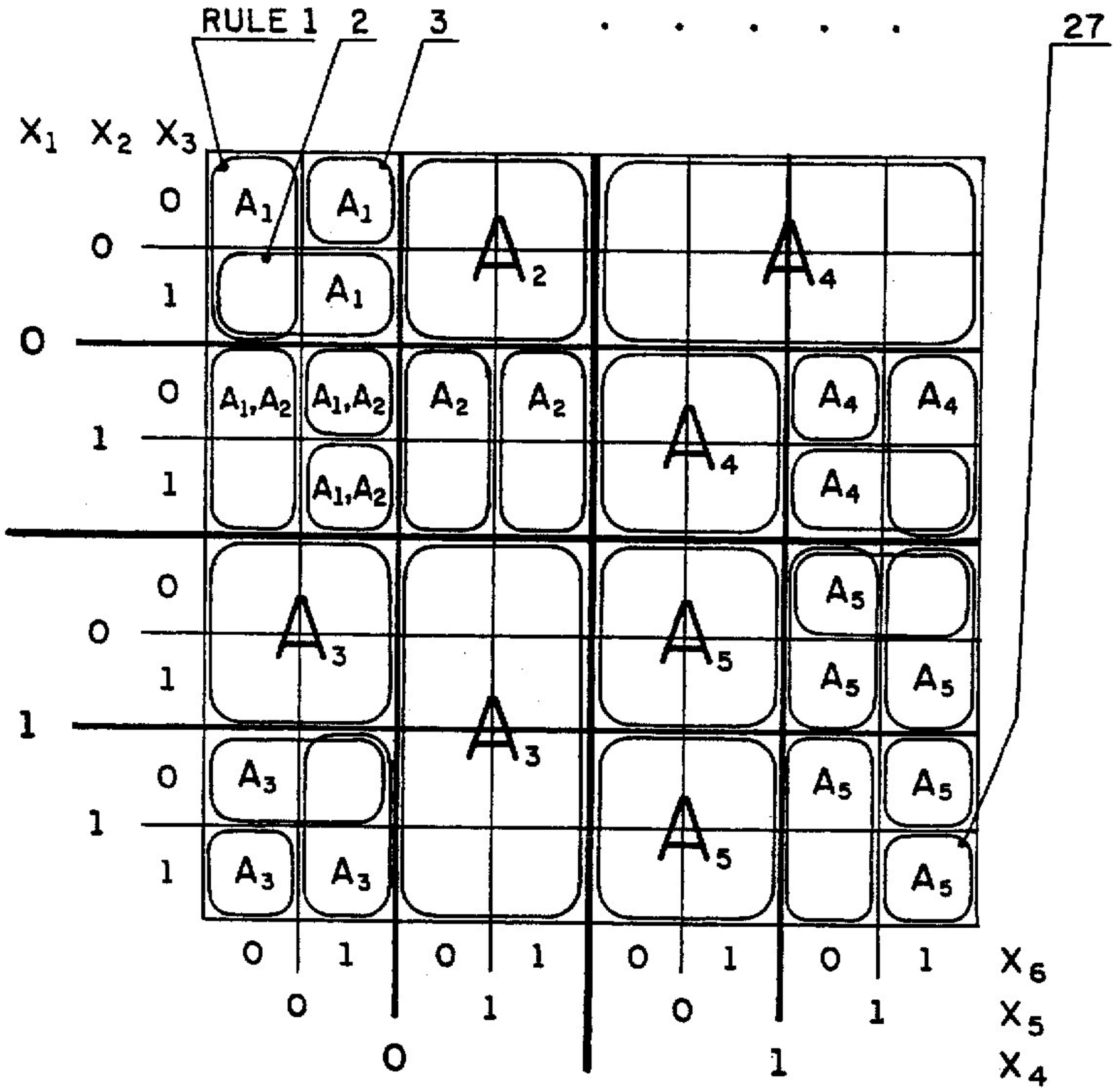
A decision diagram, for a given decision table, is constructed by locating in the diagram (representing the space **E** of test outcomes) the complexes which correspond to the condition part of every rule, and marking them by actions specified in the action part.

A complex (or a cell) marked by action A is called in the sequel a complex (or cell) of class A. Fig. 3 and 4 present the decision diagrams representing decision tables in Fig. 1 and 2, respectively. It may be a useful exercise for the reader to check the correspondence between the rules in the decision tables, and corresponding complexes in the decision diagrams.

In this paper, decision diagrams are used, both, as a conceptual geometrical model for describing algorithms, and as a visual aid for solving problems. A significant advantage of decision diagrams lies in the fact, that it is much easier (for humans) to see differences and similarities between geometrical configurations, than between strings of numbers or symbols.

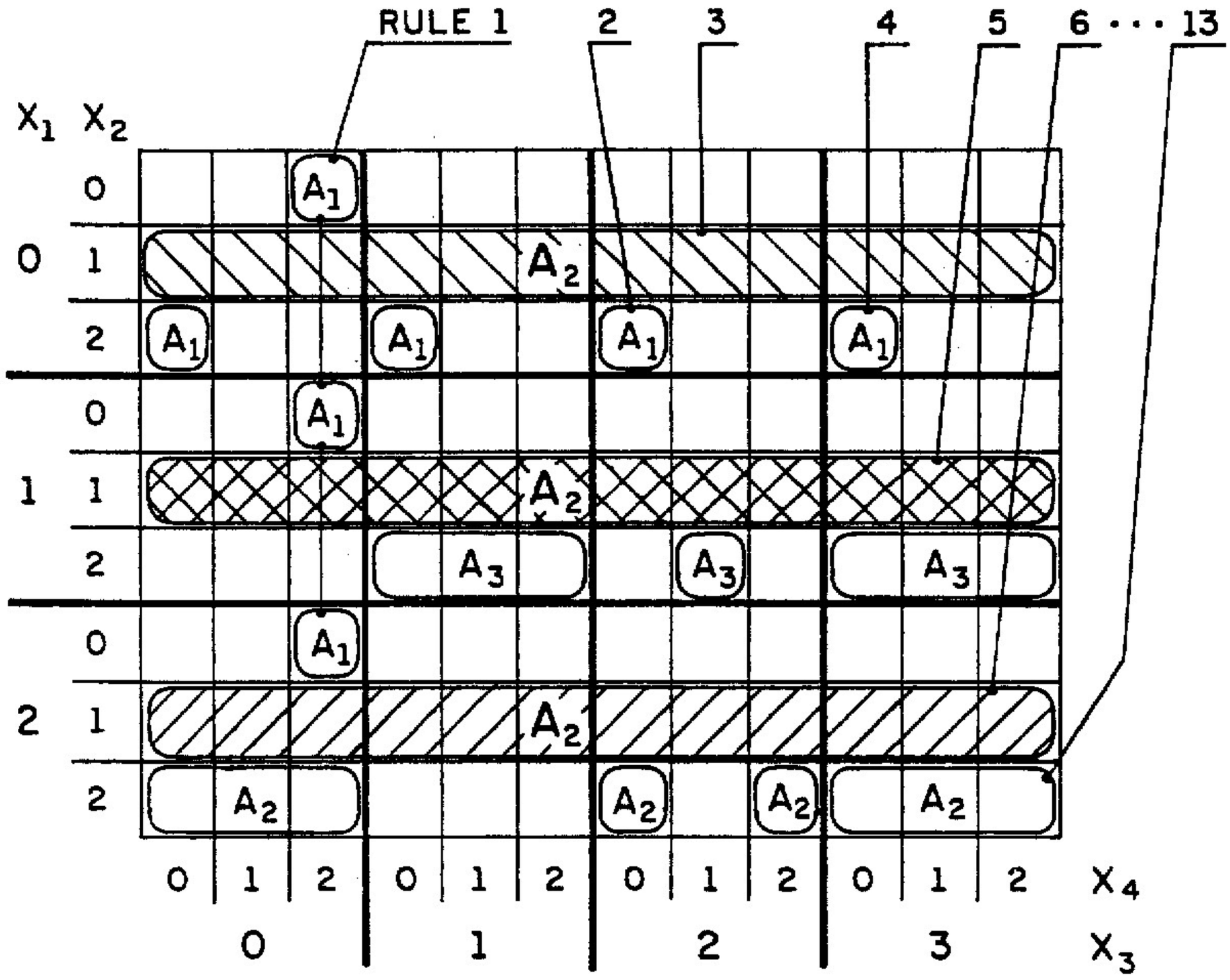
---

* In the case of binary tests (i.e., when $a_i \in [0,1]$), a complex of 2^k cells corresponds to a k-cube (a subset of 2^k vertices of an n-dimensional hypercube).



Decision diagram representing decision table in Fig. 1.

Figure 3.



Empty cells correspond to
ELSE conditions.

Decision diagram representing
decision table in Fig. 2.

Figure 4.

A description of algorithms in terms of geometrical constructs (which can be visualized) has therefore a great appeal - both for scientific communication and education.

In the past, many authors used the concept of an n-dimensional hypercube and its subsets, k-cubes, for representing an event space and logical products, respectively. A hypercube, however, can be directly visualized only when there are not more than 3 variables; when there are more than 3 variables, it rapidly looses its value as a geometrical model. When the variables can take more than 2 values (as in our case) the concept of a hypercube is even less adequate.

Although a form of diagrams with binary tests (Karnaugh maps) has been used in the past for solving problems related to limited entry decision tables, this is the first paper, to the author's knowledge, which demonstrates useulness of diagrams for extended entry decision tables, and uses them systematically as a conceptual model for presenting and analyzing algorithms. The paper also demonstrates that decision diagrams are a useful practical tool (when the use of a computer is not necessary) for directly solving various problems related to decision tables, such as testing decision tables for redundancy, consistency and completeness, optimizing decision tables, and quickly converting them to optimal (or near-optimal) decision trees.

Chapter 2 describes the use of decision diagrams in designing and optimizing decision tables, and Chapter 3 gives a theoretical analysis of the problem of converting decision tables to (space or time) optimal decision trees, and describes two first degree conversion algorithms. Chapter 3 also demonstrates a need, in some cases, for conversion algorithms of higher degree than first, and shows that such algorithms can be easily obtained from the first degree algorithms.

II. USE OF DECISION DIAGRAMS IN DESIGNING DECISION TABLES

2.1 Testing decision tables for redundancy, consistency and completeness.

A well designed decision table should be non-redundant, consistent and

complete.* These properties can be easily tested once a decision diagram has been constructed for the given decision table. The redundancy occurs, if the decision diagram contains complexes of the same action class, which have a non-empty intersection. For example, in Fig. 3, complexes representing rule 1 and 2 of action class A_1, intersect, and therefore, the decision table in Fig. 1 is redundant. If intersecting complexes are of different action classes, then the decision table is inconsistent. The decision table is complete, if every cell of the decision diagram belongs to (is covered by) some complex. We can see in Fig. 3 that the decision table from Fig. 1 is redundant, consistent and complete; and in Fig. 4, that the decision table from Fig. 2, is irredundant, consistent and complete (the table would be incomplete if there were no ELSE rule).

2.2 Optimization of a decision table

It is usually desirable that a decision table contains the minimum number of rules, which is sufficient for specifying the given decision problem and preserving the requirements of non-redundancy**, consistency and completeness. In a decision diagram, a reduction in the number of rules occurs when two or more complexes of the same action class are merged (or rearranged) into a smaller number of complexes. The theoretical basis for merging complexes is given by the simplification rule:

$$L[x_i=0] \vee L[x_i=1] \vee \ldots \vee L[x_i=d_i-1] = L \qquad (2)$$

where L is a term,

$L[x_i=a]$ is a logical product of L with condition $[x_1=a]$,

$\{0,1,\ldots,d_i-1\}$ is the set of all possible outcomes of test x_i, i.e., D_i.

---

* A decision table is
- redundant, if there is a combination of test outcomes which satisfied the condition part of more than one rule with the same action part;
- inconsistent, if there is a situation as above but when the action parts are different;
- complete, if it contains a rule for any sequence of test outcomes.

** If one permits a redundancy, the number of rules can sometimes be further reduced.

The rule (2) applied to a decision diagram says that if complexes of the same action class differ in the outcome of only one test, and the test takes on all possible values in these complexes, then the complexes can be merged into one complex not involving this test at all. If certain combinations of test outcomes can never occur (are 'DON'T CARE-s'), then cells corresponding to them (empty cells in Fig. 4) can be included in any complex if this can help to merge complexes.

Let $\{A_i\}$, $i=1,2,\ldots,m$, denote the set of all action classes, and E_i - the set of all cells of action class A_i in a given decision diagram.

A <u>cover</u> $C(A_j)$ of <u>action class</u> A_j is a set, $\{C_k\}$, of complexes, whose union includes (covers) set E_j and does not cover any cells of other action classes:

$$E_j \subseteq \bigcup_k C_k \subseteq \mathbf{E} \setminus \bigcup_{\substack{i=1 \\ i \neq j}}^{m} E_i \quad . \tag{3}$$

If all complexes in $C(A_j)$ are pairwise disjoint, then the cover is called a <u>disjoint cover of class</u> A_j.

If covers $C(A_i)$ of classes A_i, $i=1, 2, \ldots, m$, have the property that any two complexes from any two such covers, respectively, do not intersect, then the union of such covers is called a <u>cover of the decision table</u>. If in a cover of a decision table, the covers $C(A_i)$ are disjoint, then the cover is called a <u>disjoint cover of the decision table</u>.

It is easy to see that any decision table which satisfies conditions of non-redundancy, consistency and completeness defines a disjoint cover of the corresponding decision diagram. The following concept is basic to the contents of the paper:

<u>Definition 1</u>. An <u>optimal disjoint cover</u> of a decision diagram is defined as a disjoint cover, which has the smallest number of complexes, and, in case of tie, includes complexes of larger size (i.e., their union covers more cells) among other disjoint covers of the diagram.

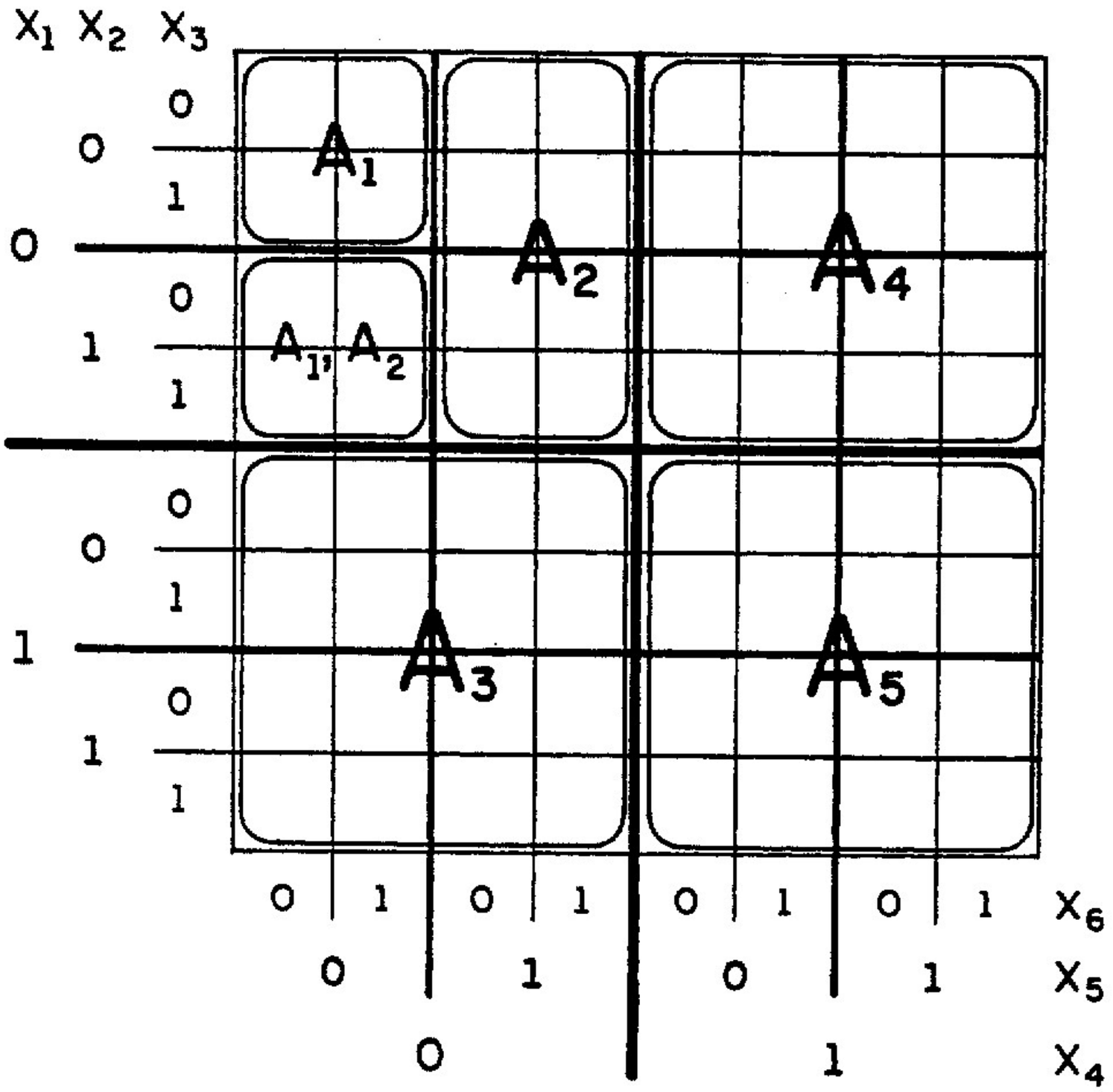
The importance of the optimal disjoint cover (called, from now on, the optimal cover) stems from the fact that it corresponds to a decision table with the smallest number of rules. If there are two or more such tables, the optimal cover corresponds to the table which has more dashes ('-') in the entries (i.e. the condition parts of the rules involve fewer specified test outomes). The decision table corresponding to the optimal cover is called the optimal decision table.

Thus, determining the optimal decision table is equivalent to determining the optimal cover of a decision diagram. It should be noted that there can be, in general, more than one optimal cover, and, therefore, more than one optimal decision table defining the same decision process. Fig. 5 and Fig. 7 present optimal covers for decision diagrams in Fig. 3 and 4, respectively. Fig. 6 and 8 show the optimal decision tables corresponding to these covers.

Determining the optimal cover of a decision diagram is similar to the process of minimizing a Boolean function in a Karnaugh map, although there are differences:

1. all complexes must be pairwise disjoint,
2. the cover involves a family of covers, one cover for each action class (unlike in Boolean minimization, when there is only one cover to be constructed)
3. the process is done in non-binary event space (assuming extended entry decision tables).

For commonly occuring decision tables (which rarely involve more than 6-7 binary or 'few-valued' tests (see, e.g., [5]) optimal covers of the corresponding decision diagrams can often be found just by visual inspection of the diagram and the application of the rule for recognition of complexes given in [1] (one can also develop his/her own recognition rule, since complexes have certain easily detectable regularities).



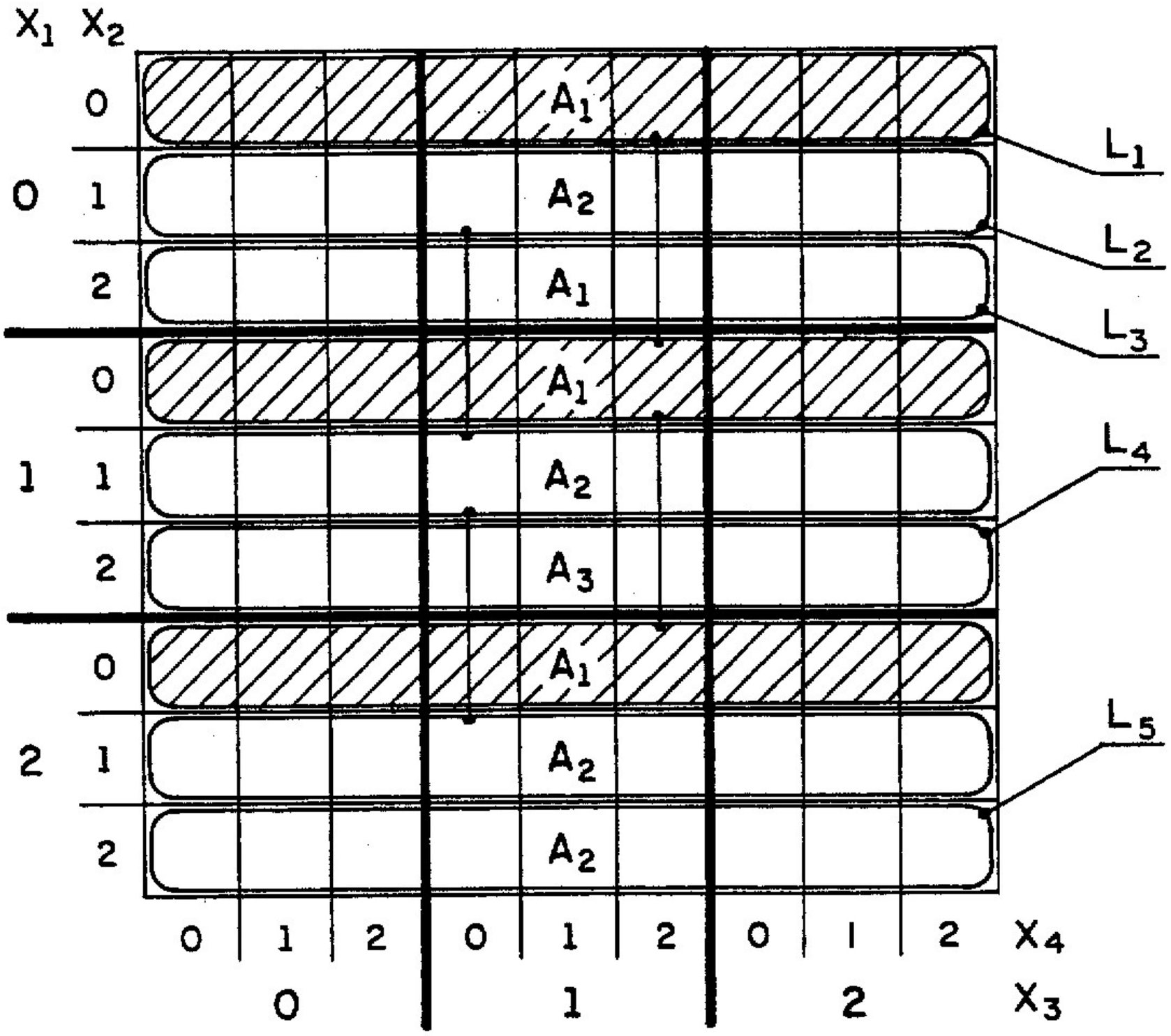
Optimal cover of decision diagram in Fig. 3.

Figure 5.

Tests	x_1	0	0	0	1	0	1
	x_2	0	1	–	–	–	–
	x_3	–	–	–	–	–	–
	x_4	0	0	0	0	1	1
	x_5	0	0	1	–	–	–
	x_6	–	–	–	–	–	–
Actions	A_1	1	1				
	A_2		1	1			
	A_3				1		
	A_4					1	
	A_5						1

Optimal decision table corresponding to the cover in Fig. 5.

Figure 6.



Optimal cover of decision diagram in Fig. 4.

Figure 7.

Tests	x_1	-	0	-	2	1
	x_2	0	2	1	2	2
Actions	A_1	1	1			
	A_2			1	1	
	A_3					1

Optimal decision table corresponding to the cover in Fig. 7.

Fig. 8

When a decision table is very large (say, more than 8 'few-valued' tests and 50 rules) it is necessary to use a computer program implementing a systematic method of decision table optimization.

Since the optimization of a decision table is equivalent to solving a special case of the general covering problem [9], any of the known covering algorithms can be adopted here. An example of a systematic method for optimizing limited entry decision tables is described in [19], which adopts the Quine-McCluskey algorithm for generating prime implicants.

Another example of a systematic method is the quasi-optimal covering algorithm A^q, described in [9] and [11]. This algorithm works differently than usual covering algorithms which first generate all candidate complexes and then select a cover: namely, it generates, at each step, only a specially chosen subset of candidate complexes, and then selects from it the 'best' complex. Due to its efficiency and generality, the algorithm A^q can be applied to very complex covering problems, involving many (easily more than 15) binary or many-valued variables. The algorithm is implemented in computer program AQVAL/1 [8].

In the sequel, the generation of the optimal cover of a decision diagram (or just a single action class) is assumed to be done by some adopted procedure.

Sometimes decision tables are constructed with an assumption that individual rules are tested in order from the left to the right, and the first rule satisfied evokes the corresponding action. This assumption usually leads to a simpler decision table, because the condition part of a rule may intersect with the condition part of any of the preceding rules. Decision diagrams can be easily used to optimize such ordered decision tables. First, construct the optimal cover of the first action class (the same way as in the 'unordered' table). Next, when constructing optimal covers of subsequent action classes, the cells covered by the preceding covers are treated as "DON'T CARE-s whenever it can simplify the cover.

III. CONVERTING DECISION TABLES INTO OPTIMAL DECISION TREES

3.1 General remarks, previous work

It is an interesting aspect of human versus machine psychology, that one prefers a decision table in order to formalize a decision process, but when one wants to program it, a decision tree is preferable. In such a tree, the nodes are labeled by individual tests, the branches by test outcomes, and the leaves by action classes. Many methods (e.g., 6, 14, 17, 1, 16, 22, 21, 2, 14, 4, 19, 18) have been devised for converting a decision table (in most cases only limited entry) to a decision tree which is optimal (or near-optimal) according to some criterion. Usually, two criteria of optimality are considered:

1. that the tree has the minimum number of nodes (such a tree corresponds to the space optimal program, i.e., a program with the minimum memory requirement), or
2. that the tree corresponds to the time optimal program (i.e., program whose average execution time is minimum).

After an early rule mark technique proposed by King [6], most subsequently developed methods (e.g., 14, 1, 16, 22, 21, 14, 4) apply the principle of decomposition. Tests to be assigned to the nodes of the tree are selected one by one, according to some heuristic criterion. Decomposition methods are computationally very efficient, but may fail to produce the strictly optimal trees.

Other approaches include a branch-and-bound algorithm by Reinwald and Soland [17], and dynamic programming methods by Bayes [2], and by Schumacher and Sevcik [18]. These algorithms always guarantee the optimal solution, but require exhaustive searches through large spaces of trees. For example, the storage requirement and the execution time of the dynamic programming method of Schumacher and Sevcik [18] both grow with the number of binary tests in proportion to 3^n. Consequently, the method becomes practically unacceptable when the number of binary tests exceeds 10-12. In the case of the branch-and-bound

algorithm [17], the situation is even worse (as estimated in [18], the algorithm may require several hours even when n=6). Since the computational cost of table conversion should be included in the total cost of tree implementation, it can happen that a suboptimal tree (obtained by an efficient decomposition method) is 'more optimal' than the theoretically optimal tree.

Most of the developed methods are strictly computer oriented, and give little insight to the nature of the difficulty of the conversion problem; thus, they leave the practitioners coping with the variety of practical conversion problems strongly dependent on the availability of computer programs and forced to accept the assumptions underlying the programs.

In this paper we analyze the conversion problem of limited and extended entry decision tables, and try to provide a clear graphical illustration of the difficulties by using decision diagrams. Throughout the paper we concentrate on the simpler problem of space optimal conversion, and then show how the results can be extended to time optimal conversion.

The conversion algorithms described here are basically decomposition algorithms. We show, however, that by extending their degree (def. 5), one can obtain a spectrum of algorithms which differ in the trade-off between the computational efficiency and the degree of guarantee of optimality. When the algorithms do not guarantee the optimum, they give a measure of the maximum possible distance to the optimum.

3.2 Problem analysis

CASE 1: Conversion of limited entry decision tables to space optimal decision trees

First, we will describe a procedure for converting a decision table, by means of a decision diagram, to an arbitrary decision tree equivalent to it.

A test, say x_i, is selected and assigned to the root of the tree under construction. The left and the right branches of the root are assigned outcomes of the test, 0 and 1, respectively. The branches are put into correspondence with parts (subdiagrams) of the decision diagram, defined by conditions $[x_i=0]$ and $[x_i=1]$, or, briefly*, with subdiagrams $D(x_i=0)$ and $D(x_i=1)$, respectively. Next, a test x_{j1}, is selected for the left, and a test x_{j2} for the right descendant of the root. The branches of the node x_{j1} are put into correspondence with subdiagrams $D(x_i=0, x_{j1}=0)$ and $D(x_i=0, x_{j1}=1)$. A similar correspondence is established for branches from node x_{j2}. It is clear now, that selection of a test for a node always implies a partitioning of the corresponding diagram into smaller parts. For the next test selection these parts can be considered as independent diagrams (although, they may consist of geometrically separate collections of cells).

If at any step of assigning a test to a node, a subdiagram corresponding to a branch of this node consists of cells of only one action class, say A, (and possibly DON'T CAREs), then this branch ends with a leaf, which is assigned class name A. The tree is completed when there are no subdiagrams with cells of different action classes.

It is obvious that if we construct this way all possible trees equivalent to the given decision diagram, then we can select from them the optimal tree. Since the number of possible trees grows very rapidly** with the number of available tests, this method of finding the optimal tree is clearly unacceptable, except for trivial cases.

---

*A part of decision diagram defined by a term $[x_{i1}=a_1][x_{i2}=a_2]\ldots[x_{ik}=a_k]$ is denoted $D(x_{i1}=a_1, x_{i2}=a_2, \ldots, x_{ik}=a_k)$.

**For n k-ary tests, k=2,3,..., the maximum number of trees is $\prod_{i=0}^{n-1}(n-i)^{k^i}$ (we have n choices for the root, n-1 choices for k descendents of the root, n-2 choices for k^2 descendants of the descendants, etc.)

How can we develop a criterion which guides the test selection at each step in such a way that the resulting tree will be optimal (or at least near optimal)? This question is the subject of the remaining part of the section.

In a binary tree, the number of leaves, l, is uniquely defined by the number of nodes, v:

$$l = v + 1 \tag{4}$$

Consequently, minimizing the number of nodes is equivalent to minimizing the number of leaves. In our decision trees, the leaves correspond to action classes. Therefore, if a decision table has m different action classes and an equivalent tree has m leaves, then the tree is, obviously, optimal. A decision tree can be non-optimal only when the tree has more than one leaf marked by the same action class.

Each leaf (branch) of a tree is defined by a sequence of outcomes of tests assigned to the nodes on the path from the root to the leaf (branch). Suppose, that these outcomes for some leaf (branch) are $a_1, a_2, \ldots, a_k$, and the corresponding tests are $x_1, x_2, \ldots, x_k$. Then the leaf (branch) defines a term

$$[x_1=a_1][x_2=a_2] \ \ldots \ [x_k=a_k] \tag{5}$$

(in a binary tree $a_i \in \{0,1\}$) and, also, a complex in a decision diagram.

Suppose, that in a given decision diagram, there are k cells of action class A. If these cells are treated separately, then there will be k leaves representing them in a decision tree. If, however, they can be put into one complex, then all of them can (potentially) be represented by one leaf. Consequently, the problem of minimizing the number of nodes in the decision tree is related to the problem of representing action classes by the minimum number of complexes, i.e., the problem of constructing the optimal cover of a decision diagram. The following theorem gives a more precise meaning of this relation:

Theorem 1: If the optimal cover of a decision diagram has s complexes, then the optimal decision tree equivalent to the diagram has at least s-1 nodes.

Proof:

According to def. 1, the optimal cover has the smallest possible number of complexes needed to cover all cells of every action class. Since leaves correspond to complexes, then any tree equivalent to the decision diagram, including the optimal one, can not have less than s leaves, and, because of (4), less than s-1 nodes. ■

Theorem 1 gives a better lower bound on the number of nodes in the optimal decision tree than value m-1 (m-number of action classes), but requires a construction of the optimal cover. It also implies that if a tree can be constructed with s-1 nodes, then the tree is optimal. Theorem 1 also indicates, that the optimal tree may have more leaves than the number of complexes in the optimal cover. The reason for this is that each node in a binary tree always has 2 branches, which correspond to 2 complexes. Therefore, the number of nodes in a tree representing a cover depends not only on the number of complexes in a cover, but also on the relationships existing among complexes.

Let R, called a reference set, denote (initially) the optimal cover of a given decision diagram. As was mentioned before, selecting a test for a node implies a partitioning of the diagram into 2 parts. Such a partitioning may break one or more complexes in R into 2 (sub)complexes. If only 1 complex is broken, then instead of 1 leaf (which could potentially represent this complex in the tree), there will have to be at least 2 leaves representing this complex in the final decision tree ('at least', because subsequent partitions may break this complex again). Thus, by selecting the given test, 1 additional leaf (and, therefore, 1 additional node) is added to the tree, above the necessary minimum. If R complexes are broken, then R leaves (and R nodes) are added to the tree. It is clear, that the number of complexes which are broken in the reference set by selecting a test (or, generally, a relation between the reference set and the test) is indicative of the number of nodes in the final tree. Consequently, properties of this relation can be used for constructing a test

selection criterion.

Suppose text x_i has been selected for a node, and some complexes have been broken in R. The diagram is partioned into 2 subdiagrams, and there are now 2 sets, R' and R", each being a cover of the corresponding subdiagram. The next selection of a test (for a node which ends the branch corresponding to one of the subdiagrams), can now be based on the relation between the R'(R") and each test.

If the initial R were a different optimal cover than the above, the whole process could lead to a different tree. To make our considerations general, let us then assume that R can stand for any cover (of the initial diagram or subsequent subdiagrams), and that our goal is to determine the 'quality' of a test by measuring a certain property of the relation between set R and the test. The following definition gives a more precise expression of the above concept.

Definition 2. A first degree cost estimate $\Delta(x_i)$ for test x_i, with regard to reference set R, is a function which assigns to x_i a positive integer depending on the relation between R and x_i.

The following is a definition of an important special case of first degree cost estimate.

Definition 3. A first degree cost estimate for x_i, which assigns to x_i the number of complexes broken by x_i in R, is called the Δ_o or static cost estimate, and denoted $\Delta_o(x_i)$.

A first degree cost estimate is insufficient to capture the full cost of selecting a test x_i, because once test x_i is selected, the reference set R is partitioned, and this partition may influence the choice of subsequent tests. In order to apply a "look ahead" in test selection, higher degree cost estimates are introduced.

Definition 4. The second degree cost estimate for text x_i, $\Delta^2(x_i)$, is defined:

$$\Delta^2(a_i) = \Delta(x_i) + \min_{x_L \in N(x_i)} \{\Delta(x_L)\} + \min_{x_R \in N(x_i)} \{\Delta(x_R)\} \qquad (6)$$

where $N(x_i)$ is the set of tests available for assignment to the descendants of node assigned test x_i ($N(x_i)$ is the set of all tests minus tests assigned to nodes on the path from the root to node x_i, inclusively),

x_L - a test assigned to the left descendant of node x_i,

x_R - a test assigned to the right descendant of node x_i.

Similarily, higher degree cost estimates can be defined.

Definition 5. A kth, k = 1,2,..., degree conversion algorithm is defined as an algorithm which assigns to each node of the tree the test, whose kth degree cost estimate is minimum.

Let us discuss in more detail the estimate Δ_o. This estimate is very attractive due to its simplicity. If the reference set R is the optimal cover of the diagram (subdiagram), $\Delta_o(x_i)$ specifies the minimum number of nodes which are added to the final tree, above the lower bound defined by the cardinality of R, if any of the parts of complexes broken by x_i cannot be (or are not) merged later into a larger complex. If, in addition, complexes broken by selecting x_i are not broken again in the consecutive steps of test selection, then $\Delta(x_o)$ gives the exact number of added nodes to the final tree.

If a complex has only 2 cells, it can be broken only once, and the probability that it will be broken again is 0. If it has 2^p cells, it can be broken at most 2^p-1 times (so many nodes in a tree are needed to break it into individual cells). Assuming that minimum Δ_o is the criterion for test selection, the probability that a complex of 2^p cells is broken t times ($t=0,1,..,2^p-1$) decreases rapidly with t. This is so, because in order for a complex to be broken t times, the reference set R has to include a special arrangement of at least 2t complexes*. The larger is t, the less likely is the occurrence of such an arrangement.

---

* In order that a test which breaks a given complex will be a "winner" t times (assuming no tie), there will have to exist in R at least two complexes each time, that are broken by all available tests. Thus, the total number of complexes must be at least 2t.

Since the above probability depends on the particular configuration of complexes in R, one can only say that, in general, the larger is the broken complex, the somewhat higher is the chance that its parts may be broken again during the tree construction. Note, however, that size alone will not cause a complex to be repeatedly broken.

> When the complex has 2^{n-1} cells, i.e., half of the decision diagram, it will not be broken even once, if minimum Δ_o is the criterion of test selection. This is so, because such a complex is defined by the value of only one test, and this test is the only test which does not break any complexes and therefore, it will be selected for the root of the tree.

In view of the above, a reasonable criterion for test selection is to use Δ_o as the primary criterion, and when there is a tie (when more than one test has the same value of Δ_o), to select the test which breaks smaller complexes. If there still is a tie, any test can be selected.

Definition 6. The above defined criterion for test selection is called the criterion of minimizing added leaves (MAL).

MAL can be viewed as another form of the first order cost estimate. Suppose a decision tree has been constructed using the MAL criterion. The sum of Δ_o for tests assigned to each node of the tree is called the total cost estimate, and denoted Σ_o. An important property of Σ_o is given by:

Theorem 2: A decision tree obtained using MAL criterion has no more than Σ_o nodes above the number of nodes in the optimal tree.

Proof:

The theorem is the direct consequence of theorem 1, definition 3, and the previously discussed meaning of Δ_o estimate. ■

Thus, if Σ_o is 0, the obtained tree is optimal.

If after any step of test selection, some, say t complexes and/or parts of the broken complexes are merged into one complex, the value of Σ_o should be decreased by t-1. Therefore, in order to get a 'better' tree using MAL criterion, one should check, after each test selection, whether parts of the broken complexes (possibly together with other complexes) can be merged

into larger complexes.

Another disadvantage of the MAL cost-estimate is that it assumes that in computing Δ_o the reference set is the optimal cover of the diagram (subdiagram) under consideration. If there are many optimal (or irredundant* covers), all of them may have to be checked in order to find out which one leads to the 'best' cover (assuming that $\Sigma_o \neq 0$ each time).

We will now develop another first order cost estimate which does not have the above disadvantages.

Let E denote the set of all cells of action class A in a given diagram (subdiagram). Selecting a test, say x_i, divides the diagram into 2 subdiagrams. Suppose, that by doing so, set E is split into 2 subsets, E_o and E_1. Let C(E), $C(E_o)$ and $C(E_1)$ denote optimal covers of sets E, E_o and E_1, respectively; and c, c_o, c_1 - the corresponding cardinalities of these covers. Obviously, $c_o + c_1 \geqslant c$. The difference between $c_o + c_1$ and c is computed:

$$\Delta(x_i, A) = c_o + c_1 - c \tag{7}$$

Let *K* denote all action classes whose cells are partitioned by selecting test x_i. $\Delta(x_i, A)$ is determined for each $A \in K$, and then their sum is computed:

$$\Delta_1(x_i) = \sum_{A \in K} \Delta(x_i, A) \tag{8}$$

Definition 7. $\Delta_1(x_i)$ is called Δ_1 or dynamic cost estimate for x_i.

To see that Δ_1 is a form of the first degree cost estimate, assume that the reference set R is the union of covers

$$R = C(E) \cup \bigcup_i (C(E_o^i) \cup C(E_1^i)) \tag{9}$$

where i scans partitions of set E by all tests being considered for an assignment to the given node. Δ_1 can then be viewed as a property of the relation between R and x_i. In using Δ_1 for test selection one can ignore the value c in (7), since it remains the same for each test. Only when a test is selected,

---

*A cover is irredundant if removing or decreasing any complex in it makes the resulting set not a cover.

one can compute the 'complete' value of Δ_1, which is needed for computing the total cost estimate Σ_1, as defined later.

The Δ_1 estimate does not have the previously mentioned disadvantages of Δ_o, and is clearly more precise in estimating the effect of a test selection on the final decision tree. Its computation, however, is more complex because at each step of test selection the optimal cover of (ever decreasing) sub-diagrams has to be computed (note, however, that after selecting a test, the cardinalities of $c(E_o)$ and $c(E_1)$ can be used in computing Δ_1 for test selection at the next level of the tree). A 'shortcut' in computing Δ_1 is also possible. Namely, if $\Delta_o = 0$, then, obviously, $\Delta_1 = 0$, and Δ_1 does not have to be computed in such case (for details, see Example 3 and remarks after algorithm D).

A question arises of which test to select when Δ_1 is the same for more than one test. In computing Δ_1, sizes of covers $C(E_o)$ and $C(E_1)$ were not taken into consideration.

If the cardinality of $C^i = C(E_o^i) \cup C(E_i^i)$ is the same for different values of i, (see (9)), then Δ_1 for corresponding tests are also the same. The covers C^i may, however, consist of complexes of different sizes (because of the DON'T CARE-s). The larger the complex, the smaller number of tests are involved in its expression, and the corresponding complex can be potentially represented by a leaf at a higher level of the tree. This may reduce number of nodes (see Example 3).

Consequently, a reasonable tie-breaking rule is to select the test for which C^i consists of larger complexes (i.e., the total number of cells in complexes of C^i is larger). When there is still a tie, any test can be selected.

Definition 8. The above defined criterion for test selection is called the criterion of dynamically minimizing added leaves (DMAL).

The DMAL criterion is a form of first order cost estimate (as is MAL). Assuming that Σ_1 denotes the sum of the estimate Δ_1 for all nodes in a tree, theorem 2 also holds for Σ_1.

Other first degree cost estimates

Pollack [14] describes a first order cost estimate in which complexes broken by a test are assigned weights (called 'column-counts') equal to the number of cells they consist of. The cost estimate ('dash-count') for a test is the sum of weights of complexes broken by selecting the test. (An assumption is made in [14] that each action class is represented by only one complex. Thus, the issue of alternative covers is not considered there, which strongly limits the applicability of the method).

According to the above estimate, breaking, e.g., 4 two-cell complexes is equivalent to breaking 1 eight-cell complex. Breaking 4 two-cell complexes adds 4 more nodes, while breaking 1 eight-cell complex adds only one node (although, if this complex is broken in the next test selections, 7 more nodes could potentially be added to the tree). It is assumed here, of course, that no subsequent merging of complexes is done. In view of what was said before about the fast decrease of the probability that a complex is broken a few times, such an estimate does not seem to be sufficiently justified (in fact, it is easy to find an example for which such an estimate will select a wrong test, while the simpler MAL criterion will select a right one).

(The tie-breaking criterion used in [14](called DELTA) favors an imbalance in the number and sizes of complexes on both sides of axes of a test (i.e., in parts of diagram defined by value 0 and 1 of a test). It is unclear how to justify such a criterion, and there is a simple counter-example to it.)

Alster [1] describes a first degree cost estimate where the weight given to broken complexes is 2^k, where k is the total number of reduced variables ('dashes') in all (essential) complexes in a cover of an action class (i.e., if there are, for example, 3 two-cell complexes in an action class and any one is broken by the test, then it will be given weight $2^3 = 8$). Thus, if there is only one complex in an action class, the estimate is equivalent to Pollack's dash-count

estimate, but if there are few complexes in a class, and each complex has more than one cell (i.e., has at least 1 dash), then any broken complex from this group will be given a very large weight. Here again, for reasons discussed before, such criterion seems to be not sufficiently justified, and it is easy to find a counter-example to it (for which both the MAL and the DMAL criterion select a correct test).

When there are alternative (non-essential) complexes, paper [1] advocates the creation of OR-groups. The weight of broken complexes in such a group is divided by the cardinality of the group. The aim of the measure is to take into account the fact, that the larger are OR-groups, the more likely it is that a cover exists whose complexes will not be broken by the test under consideration.

Note, that in computing Δ_1, rather than attempting to estimate (by the above or any other measure) the probability that such a cover exists, one simply directly searches for the cover. This is computationally acceptable, and at the same time the estimate is more precise.

CASE 2: Conversion of extended entry decision tables.

In an extended entry decision table, tests may have an arbitrary number of outcomes. Consequently, the equivalent trees may have an arbitrary number of branches. Let us assume first, that all tests have the same number of outcomes equal to d, and the corresponding decision tree will be d-ary. In a d-ary tree, the number of leaves is

$$l = (d-1)v+1 \qquad (10)$$

where v is the number of nodes. Thus, as in the case of binary trees, a d-ary tree with the minimum number of leaves has also the minimum number of nodes. Both criteria of test selection, MAL and DMAL, can be adoped here without modification, as it is shown below.

Selecting a test corresponds now to partitioning a diagram into d parts. Consequently, if a test breaks a complex, it breaks it into d smaller complexes. This adds d-1 leaves and 1 node to the tree. Therefore, decreasing the number

of complexes which are broken by a test, decreases the number of nodes. Value d makes no difference with regard to which test should be selected. Both principles, MAL and DMAL, can be directly applied.

If, however, tests can have different numbers of outcomes, the trees with the same number of leaves can have different number of nodes. For example, Figure 9 shows a decision diagram which can be converted to 2 trees with the same number of leaves but different number of nodes, as shown in Fig. 10.

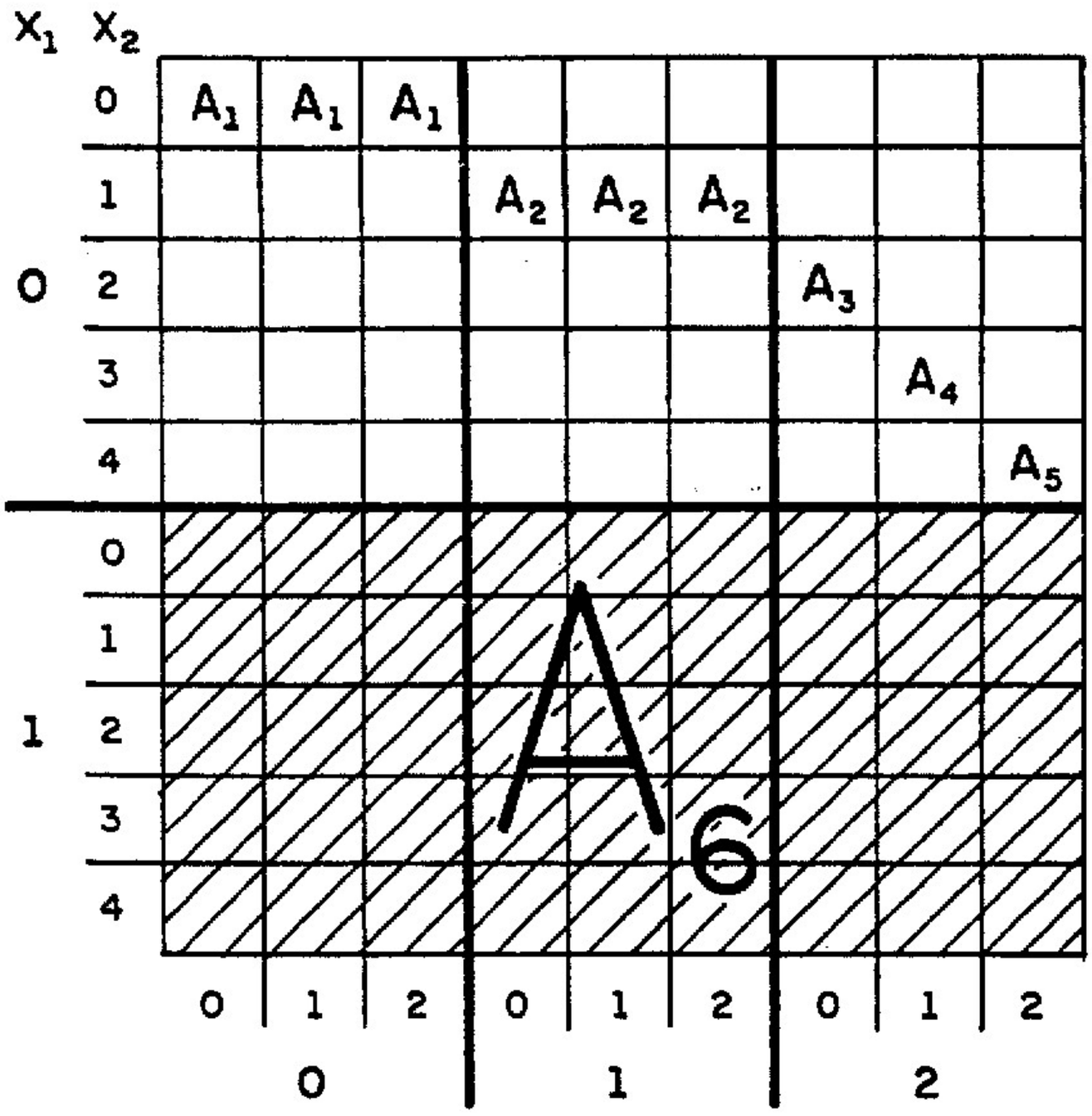
If two trees have the same number of leaves, then the tree in which tests with larger number of outcomes are assigned to nodes closer to the root will have a smaller number of nodes. Therefore, a reasonable generalization of MAL and DMAL criterion is to accept as the primary tie-breaking rule (for tests with the same value of Δ_o and Δ_1, respectively) the preference for tests with larger number of outcomes, and then, as the secondary tie-breaking rule, the one referring to the size of complexes. Thus, we have:

Definition 9. The (modified) criterion MAL {DMAL} for test selection is defined:

1. Choose the test for which Δ_0 $\{\Delta_1\}$ is smaller,
2. In case of a tie, chose the one with larger number of outcomes.
3. If there is still a tie, chose the one which partitions the diagram into parts with smaller complexes {into parts in which covers of the same class have larger complexes},
4. If there is still a tie, chose any test.

Note, that in the case when all tests have the same number of outcomes, the above defined MAL and DMAL criteria are equivalent to their previously defined form (def. 6 and 8).

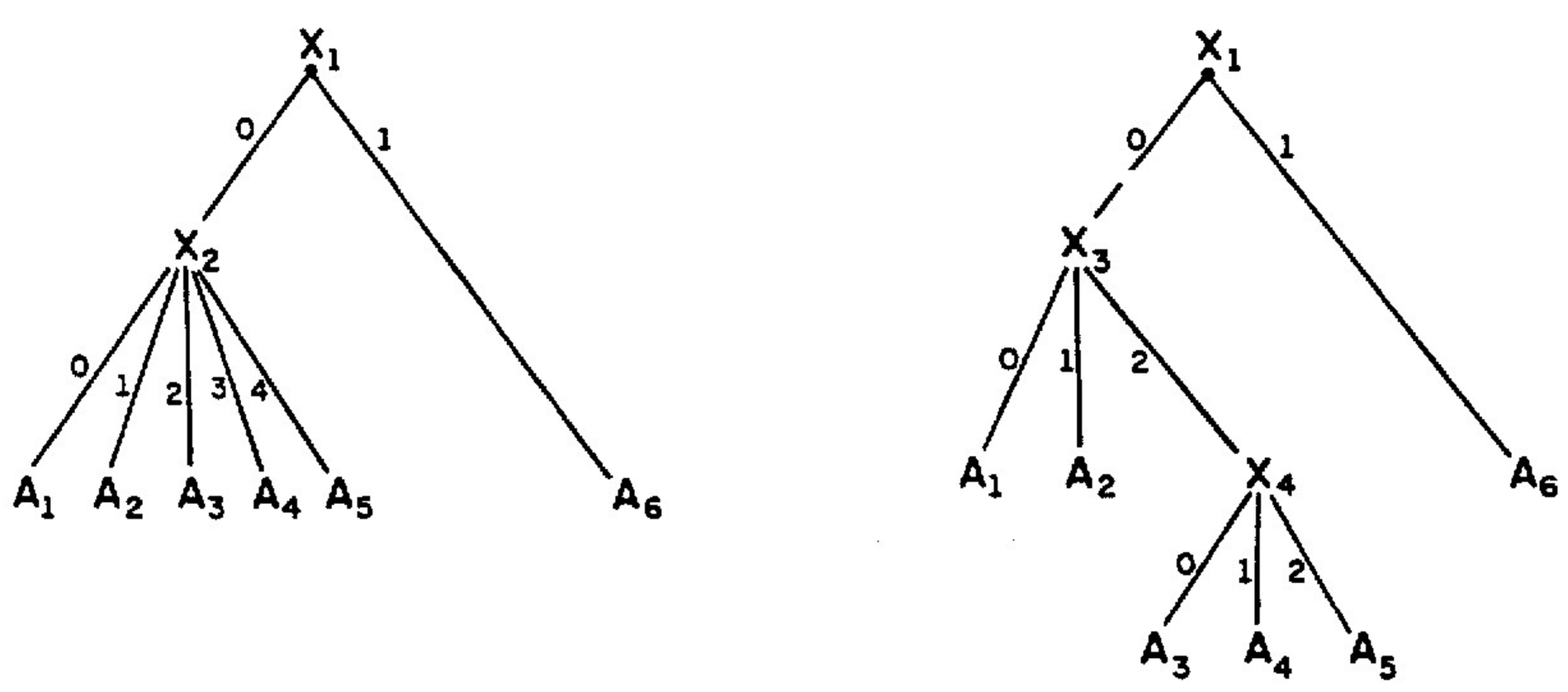
Although using the MAL or DMAL criterion for test selection will often lead to the optimal tree, in some cases the obtained tree will be sub-optimal. In such cases, a higher degree cost estimate may be needed for the "right" test selection. An example of such a case is given in the section 3.3 (Example 4).



Empty cells denote DON'T CARE-s.

A decision diagram involving tests with different number of outcomes.

Figure 9.



Two decision trees equivalent to decision diagram in Fig. 9.

Figure 10.

3.3 Algorithms and examples

The previous section described 2 criteria, MAL and DMAL, for test selection, but left unspecified the details of using them for constructing decision trees. This section describes 2 conversion algorithms, S('static cover') and D('dynamic cover') which employ the criteria MAL and DMAL, respectively. Although algorithms are described in the context of using decision diagrams, they can be directly adopted for computer implementation.

The algorithms permit someone, with practice in recognizing complexes in a diagram, to quickly and directly convert a decision diagram into an optimal or near-optimal decision tree. In the latter case, the algorithms give an estimate, Σ_o or Σ_1, respectively, of the maximum difference (in the number of nodes) between the obtained and optimal tree. The algorithms assume as given, a procedure for constructing the optimal cover of a decision diagram.

Algorithm S

The algorithm uses MAL criterion for test selection and assumes that the initial reference set R is the optimal cover of the decision diagram (or one of the alternative optimal covers, if such exist). Since a different R can produce different decision trees, in order to obtain the 'best' tree the algorithm may have to be repeated for each alternative cover (unless for some tree, the total cost estimate $\Sigma_0 = 0$).

The algorithm is recommended, when there exist only one or very few optimal (or irredundant*) covers.

Step 1: Determine the optimal cover of the decision diagram, and accept it as the reference set R. Assign the set of all tests to T. Set a printer P to indicate the root of the tree.

---

*An interesting and, to this author's knowledge, unsolved problem is whether the optimal tree can be always derived from the optimal cover (assuming that splitting complexes or joining previously split parts are the only permissible operations on the cover).

Step 2: For each test from T compute estimate Δ_o (def. 3). If for some test, x_i, $\Delta_o(x_i) = 0$, go to step 3. If for every test $\Delta_o \neq 0$, select a test according to MAL criterion (def. 9). Let x_i denote the selected test, and $0,1,\ldots,d_i-1$ be its outcomes.

Step 3: Assign x_i to node P, and outcomes of x_i, values $0,1,\ldots,d_i-1$, to branches of node P (in order from the left to the right). Split the diagram into d_i (sub)diagrams $D(x_i=0)$, $D(x_i=1),\ldots,D(x_i=d_i-1)$. Check if any of these diagrams contain a complex (or complexes) of the same action class. For each such diagram, assign the name of the action class to the end (leaf) of the branch. Put the remaining diagrams on the list L. If L is empty, then STOP.

Step 4: Apply algorithm S, starting from step 2, to each of the diagrams on the list L. Assume the following initialization for each diagram:

- P points to the node at the end of the branch corresponding to the given diagram,
- $T = T \setminus \{x_i\}$, where $\setminus$ is the set subtraction
- Merge, if possible, any complexes or parts of the broken complexes (which lie within the scope of the diagram), which are of the same action class, into larger complexes. If k complexes are merged into 1, subtract value k-1 from $\Delta_o(x_i)$. Accept the final set of complexes as the reference set R. (The above merging is not a necessary operation; if used, it can sometimes improve the final tree.) After completing the tree, compute the total cost estimate Σ_o (see theorem 2).

Example 1

Convert the decision table in Fig. 1 to a decision tree using algorithm S (Fig. 3 shows the corresponding decision diagram).

Step 1: The otpimal cover of the decision diagram is determined (Fig. 5). (Since only one complex is associated with each decision class; complexes are identified by symbols denoting classes)

$$S:=(A_1,A_2,(A_1,A_2),A_3,A_4,A_5),\ T:=(x_1,x_2,\ldots,x_6)$$

Step 2: Compute Δ_o for each $x_i \varepsilon T$:

$\Delta_o(x_1) = 0$

$\Delta_o(x_2) = 4$ (axes of x_2 cut A_2,A_3,A_4,A_5)

$\Delta_o(x_3) = 6$ (axes of x_3 cut all complexes)

$\Delta_o(x_4) = 0$

$\Delta_o(x_5) = 3$ (axes of x_5 cut A_3,A_4,A_5)

$\Delta_o(x_6) = 6$ (axes of x_6 cut all complexes).

Since $\Delta_o(x_1) = 0$, remaining values of Δ_o do not have to be computed (unless one wants to derive alternative trees; they were computed here for illustration). Test x_1 is selected.

Step 3: x_1 is assigned to the root of the tree; left and right branches of the root are assigned values 0 and 1, respectively. Split the diagram to 2 diagrams, $D(x_1=0)$ and $D(x_1=1)$. Since both diagrams contain complexes of different classes, $L:=\{D(x_1=0),\ D(x_1=1)\}$.

Step 4: Consider diagram $D(x_1=0)$ first. P points to the node which ends the branch 0 from the root. $T:=(x_2,x_3,x_4,x_5,x_6)$. The reference set $R:=(A_1,A_2,(A_1,A_2),A_4)$.

Step 2': Compute Δ_o for each $x_i \in T$:

$\Delta_o(x_2) = 2$ (axes of x_2 cut A_2 and A_4)

$\Delta_o(x_3) = 4$ (axes of x_3 cut $A_1(A_1,A_2),A_2,A_4$)

$\Delta_o(x_4) = 0$

Select x_4.

Step 3': x_4 is assigned to P, the left and the right branches are assigned 0 and 1, respectively. The diagram $D(x_1=0)$ is split into 2 new diagrams, $D(x_1=0,\ x_4=0)$ and $D(x_1=0\ ,\ x_4=1)$. Since $\Delta_o(x_4)=0$, no merging is possible. Diagram $D(x_1=0,x_4=1)$ consists of one complex A_4. Therefore, the end of branch 1 is marked by A_4.

$$L: = \{D(x_1=0,\ x_4=0)\}$$

Step 4': Apply algorithm S, starting from step 2, to diagram $D(x_1=0,\ x_4=0)$, assuming the initialization: P points to the node ending branch 0 (of node x_4); t: = (x_2,x_3,x_5,x_6), R: = $(A_1,(A_1,A_2),A_2)$

.
.
.

If one continues the algorithm, the end result will be the tree presented in Fig. 11. It is easy to check that Δ_o for each selected list was 0, therefore, the total cost estimate $\Sigma_o=0$, and the tree is optimal.

Example 2

Convert the extended entry decision table from Fig. 2 to a decision tree using algorithm S (the corresponding decision diagram is shown in Fig. 7).

Step 1: The optimal cover of the diagram is determined (Fig. 7). R: = $(L_1,\ L_2,L_3,L_4\ L_5)$, T: = (x_1,x_2,x_3,x_4), P points to the root of the tree.

Step 2: Compute Δ_o estimate for each $x_1 \in T$:

$\Delta_o(x_1) = 2$ (axes of x_1 cut complexes L_1 and L_2)

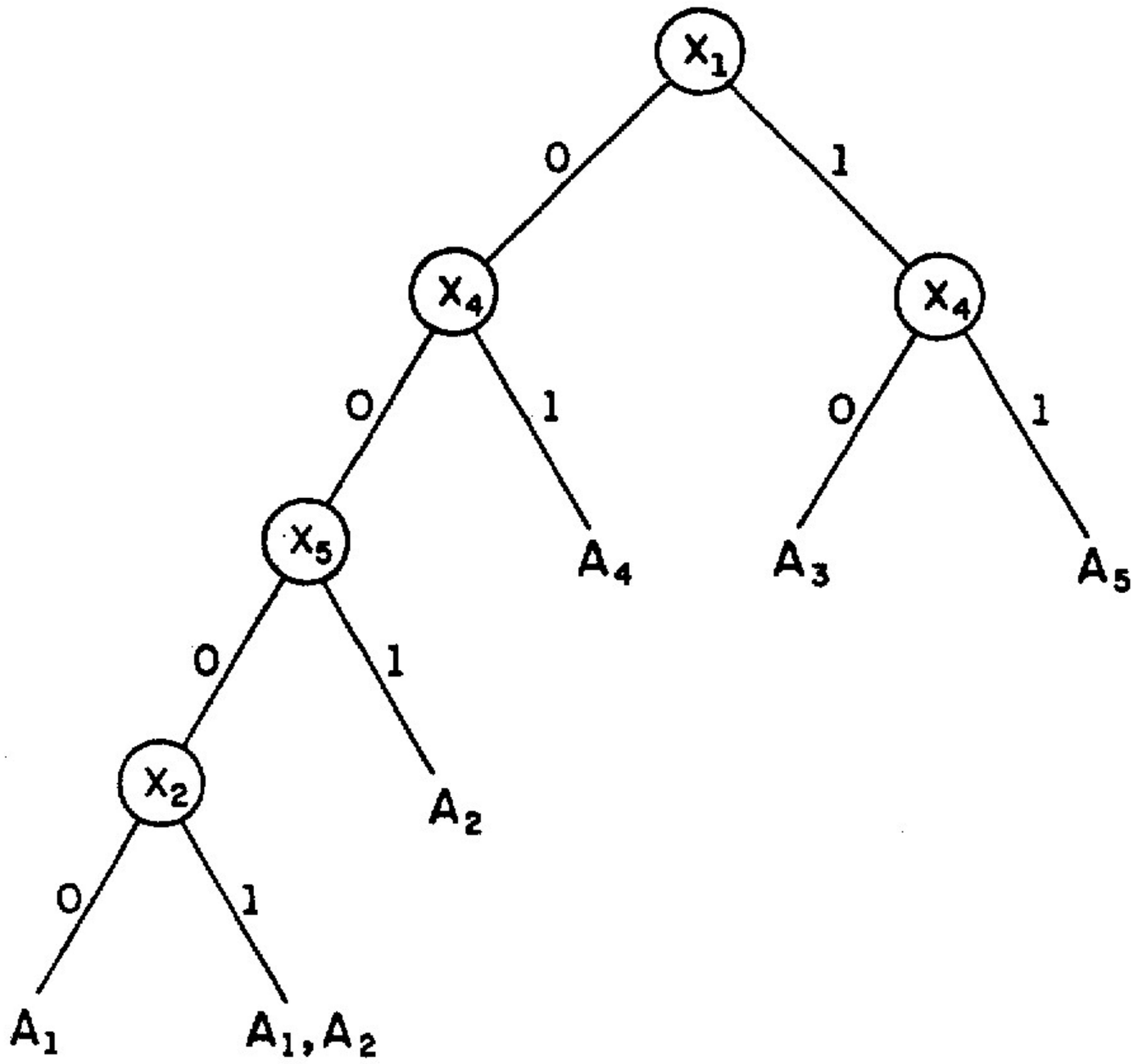
$\Delta_o(x_2) = 0$

$\Delta_o(x_3) = 5$ (axes of x_3 cut all complexes)

$\Delta_o(x_4) = 5$

Test x_2 is selected.

Step 3: x_2 is assigned to P (the root of the tree); branches from P are assigned values 0, 1, 2. The decision diagram is split into 3 (new) diagrams $D(x_2=0)$, $D(x_2=1)$ and $D(x_2=2)$. Diagram $D(x_2=0)$ consists of complex L_1 of decision class A_1, and diagram $D(x_2=1)$ of complex



A space optimal decision tree corresponding to the decision table in Fig. 1 (Example 1).

Figure 11.

L_2 of class A_2. Ends of branches 0 and 1 are marked A_1 and A_2, respectively. The list L: = $\{D(x_2=2)\}$.

Step 4: Apply the algorithm, starting from step 2, to diagram $D(x_2=2)$.

.
.
.

Continuing this process produces the tree presented in Fig. 12 Since Δ_o for each test was 0, the total cost estimate $\Sigma_o=0$ and the tree is optimal.

Algorithm D

This algorithm uses DMAL criterion for test selection. It is particularly recommended when there are quite a few choices of complexes for a cover, and, therefore, there can be a large number of irredundant (and perhaps also optimal) covers. The algorithm starts with a decision diagram in which cells of all action classes are treated as separate* (i.e., not included in any complexes).

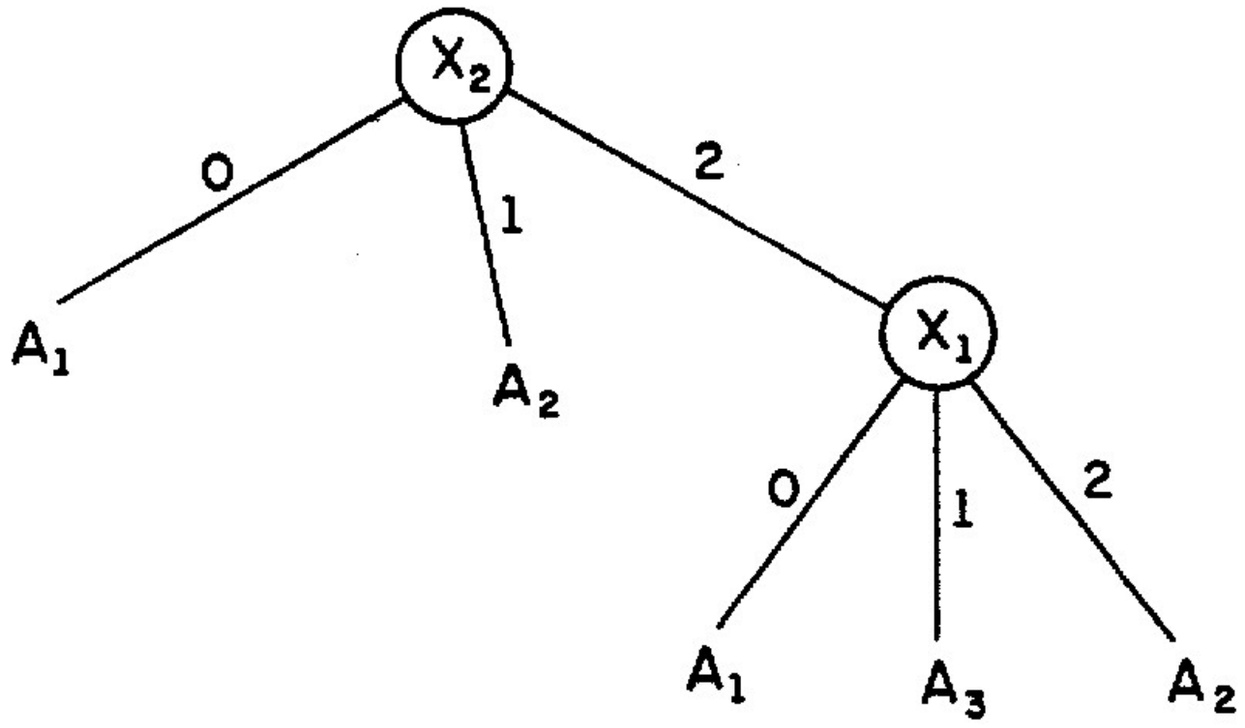
Step 1: Assign the set of all tests to T. Set P to indicate the root of the tree.

Step 2: For each $x_i \in T$, compute cost estimate Δ_1 (def. 7). Select the best test by applying DMAL criterion (def. 9).

Step 3: Assign x_i to node P, and outcomes of x_i, values $0,1,\ldots,d_i-1$, to branches of P. Split the diagram into d_i diagrams $D(x_i=0)$, $D(x_i=1)$, ..., $D(x_i=d_i-1)$. Check if any of the diagrams contain cells of only 1 action class. For each such diagram assign the name of the action class. For each such diagram assign the name of the action class to the end of the branch corresponding to the diagram. Put the remaining diagrams on the List L. If L is empty, then STOP.

---

*This condition is not necessary, if the adopted covering algorithm can find optimal covers starting with complexes rather than individual cells.



A space optimal decision tree corresponding to the decision table in Fig. 2 (Example 2).

Figure 12.

Step 4: Apply algorithm D, starting from step 2, to each of the diagrams on L. Assume the following initialization for each diagram:

- P points to the node at the end of the branch corresponding to the diagram,
- $T := T \setminus \{x_i\}$.

After completing the tree, compute Σ_1, i.e., the sum of values of Δ_1 for tests assigned to each node of the tree. Σ_1 is the maximum possible difference (in the number of nodes) between the obtained tree and the optimal tree.

A 'shortcut' in executing algorithm D is to mark (record) in the diagram (considered at a given step), the optimal covers $C(E_0)$, $C(E_1),\ldots,C(E_{d_i-1})$, which are generated at this step for computing Δ_1 (def. 7). When Δ_1 is computed in the framework of the subdiagrams (of the above diagram), Δ_o is determined with an $C(E_i)$ as the reference set. If $\Delta_o=0$, then $\Delta_1=0$. (See Example 3, step 2, for illustration.)

Example 3

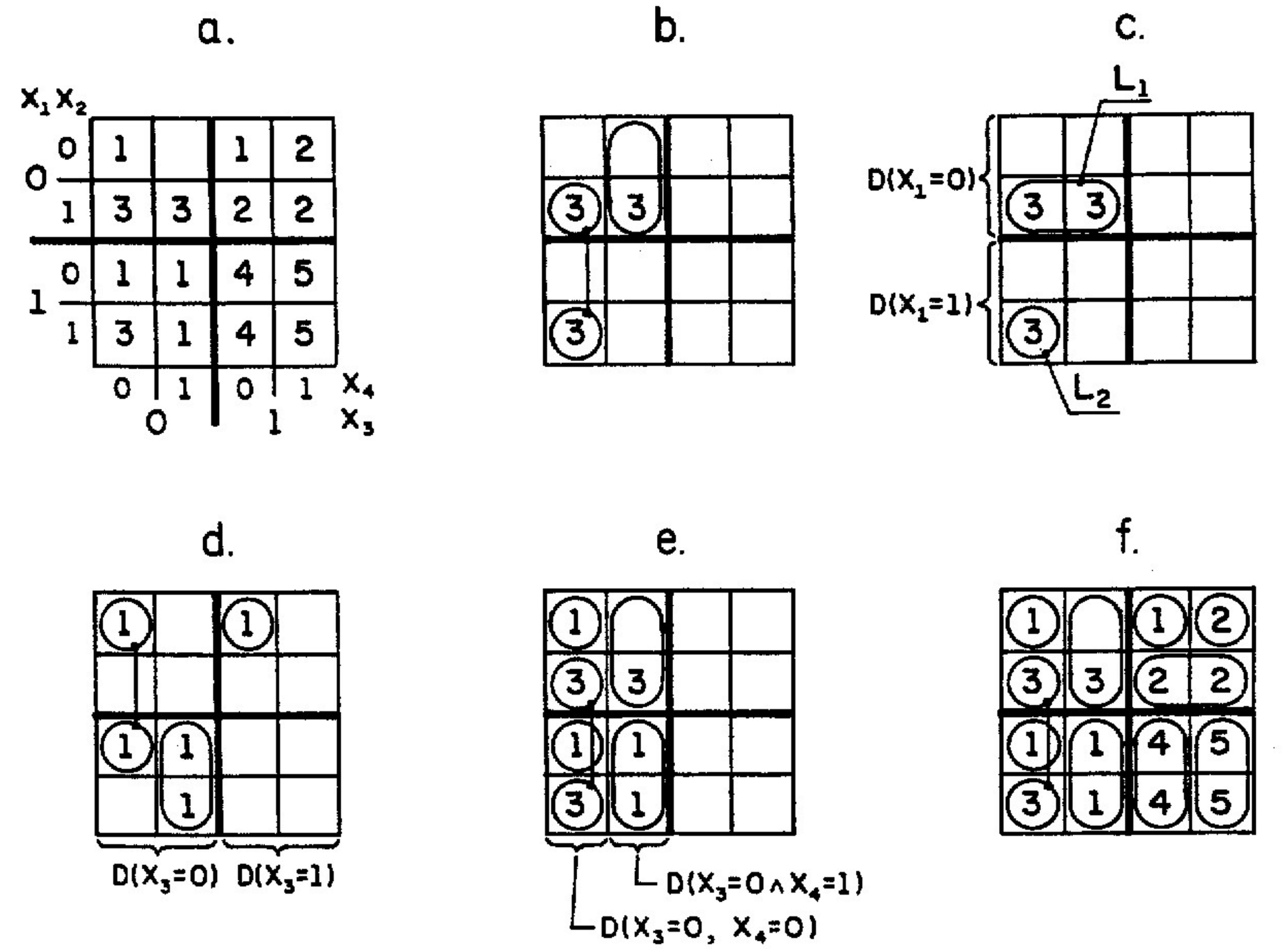
Convert the decision diagram in Fig. 13a to a decision tree using algorithm D.

Step 1: $T:=(x_1,x_2,x_3,x_4)$. P points to the root of the tree.

Step 2: Compute Δ_1 (def. 7) for each $x_i \in T$:

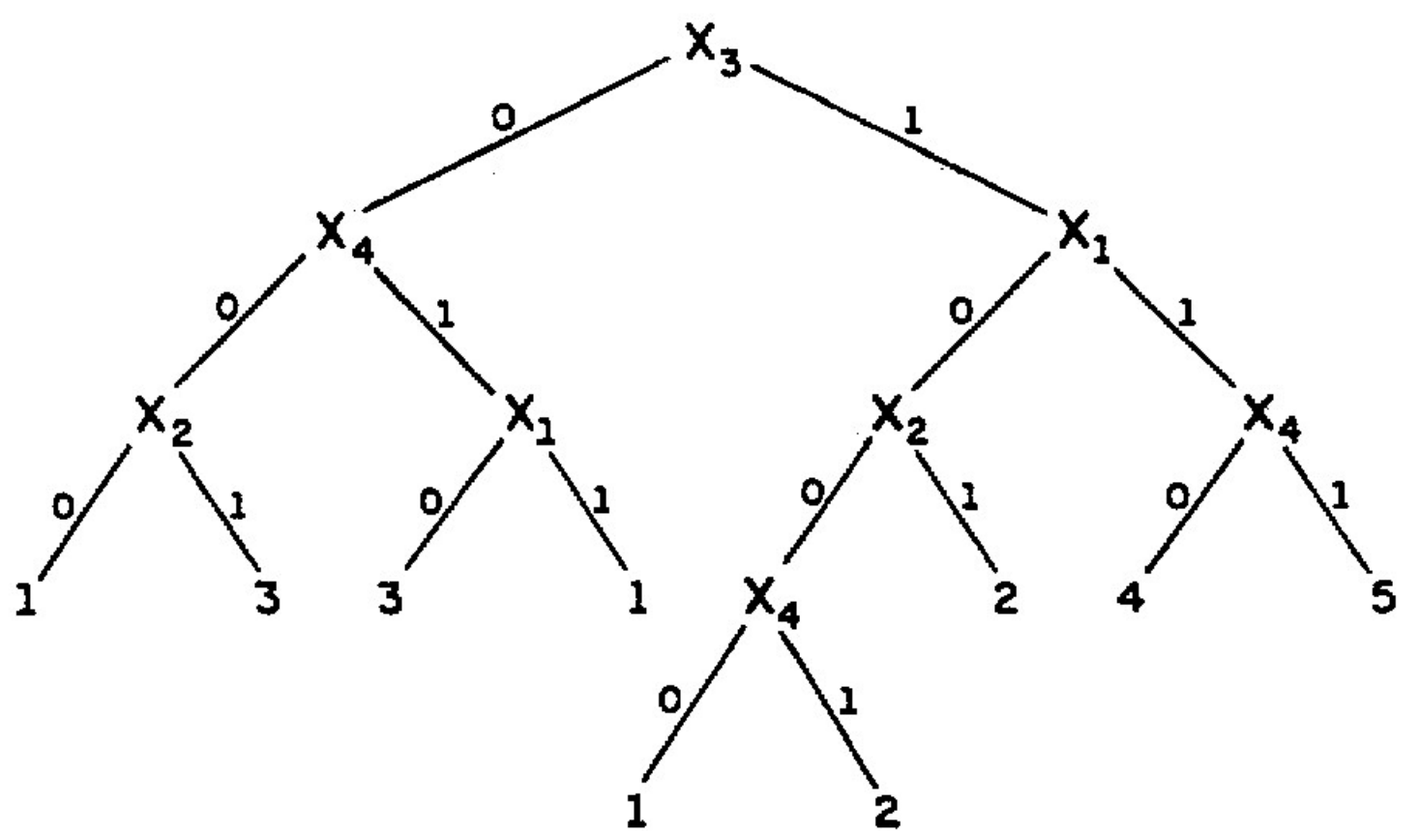
$\Delta_1(x_1)=1$

The axis of x_1 divides cells of action class 1 and 3. (Cells of classes 2,4 and 5 are on one side of the axis. The optimal cover of cells of class 1, in the original diagram, consists of 3 complexes. The optimal cover of cells in class 1 in diagram $D(x_1=0)$ consists of 1 complex, and in $D(x_1=1)$ of 2 complexes. Thus, $\Delta(x_1$, class 1) = (1 + 2) - 3 = 0. The optimal cover of class 3 in the original diagram consists of 2 complexes (Fig. 13b). The optimal cover of class 3 in $D(x_1=0)$ has 1 complex (L_1), and in $D(x_1=1)$ also 1 complex (L_2); see Fig. 13c. Thus, $\Delta_1(x_i$, class 3)= (1+1) -2 =0, and, finally, $\Delta_1(x_1) = \Delta_1(x_1$,class 1) + $\Delta_1(x_1$,class3) = 0.



Decision diagrams for example 3.

Figure 13.



Optimal decision tree for decision diagram in Fig. 13a.

Figure 14.

Note that the cover of class 3 in Fig. 13b has larger complexes than the cover of this class in Fig. 13c. This is the reason for selecting later test x_4, rather than x_1, although for both $\Delta_1=0$ (see criterion 3 for DMAL in def. 9).

$\Delta_1(x_2) = 2$ (because of action classes 4 and 5)

$\Delta_1(x_3) = 0$

$\Delta_1(x_4) = 0$

We have a tie for x_1, x_3, x_4. Condition 3 of DMAL eliminates test x_1. From remaining x_3 and x_4, x_3 is chosen. To apply the 'shortcut', the optimal cover of class 1 in diagrams $D(x_3=0)$ and $D(x_3=1)$ is marked (Fig. 13d).

If the above cover is taken now as the reference set R, then $\Delta_o(x_4) = 0$, in the framework of $D(x_3=0)$. This implies that $\Delta_1(x_4,$ class 1) = 0, and, therefore constructing optimal covers for class 1 in $D(x_3=0), x_4=0)$ and $D(x_3=0, x_4=1)$ is avoided.

Step 3: x_3 is assigned to P. The diagram is split into 2 diagrams, $D(x_3=0)$ and $D(x_3=1)$. $T:=(x_1, x_2, x_4)$.

Step 4: Algorithm D is now applied separately to diagrams $D(x_3=0)$ and $D(x_3=1)$. For the left offspring of node x_3, test x_4 is chosen. Fig. 13e shows the optimal covers of classes 1 and 3 in subdiagrams $D(x_3=0, x_4=0)$ and $D(x_3=0, x_4=0)$.

.

.

Continuing the algorithm leads to the decision tree in Fig. 14. The tree corresponds to the (optimal) cover shown in Fig. 13f.

The total cost estimate $\Sigma_1=0$, therefore the tree is optimal.

3.4 A comparison of algorithms S and D. Need for higher degree algorithms in some cases.

Algorithm S starts with constructing the optimal cover of the given decision diagram. The need for applying a covering algorithm arises again when come complexes are broken and there is a possibility that their parts and

possibly some other complexes of the same action class, could be merged into larger complexes. In algorithm D, the covering algorithm is applied at every step of test selection, although, for ever decreasing (sub)diagrams, and only for classes which are divided by a test (and except when the 'shortcut' is possible). In total, algorithm D requires more applications of the covering procedure, and, as result, takes more computation time.

On the other hand, estimate Δ_1 is more precise than Δ_o, and, therefore, the algorithm D may produce a 'better' tree than algorithm S, when there are many irredundant covers possible for a given decision diagram.

Both algorithms are first degree algorithms, and, as the following example shows, may fail to construct the optimal decision tree.

Example 4

Fig. 15 gives an example of a decision diagram for which algorithms S and D (or any other first degree algorithm) fail to produce the optimal decision tree. (Example was constructed by Yasui [22]).

Let us compute estimates Δ_o and Δ_1 for all tests:

$\Delta_o(x_1) = 2$ $\qquad$ $\Delta_1(x_1) = 2$

$\Delta_o(x_2) = 2$ $\qquad$ $\Delta_2(x_2) = 2$

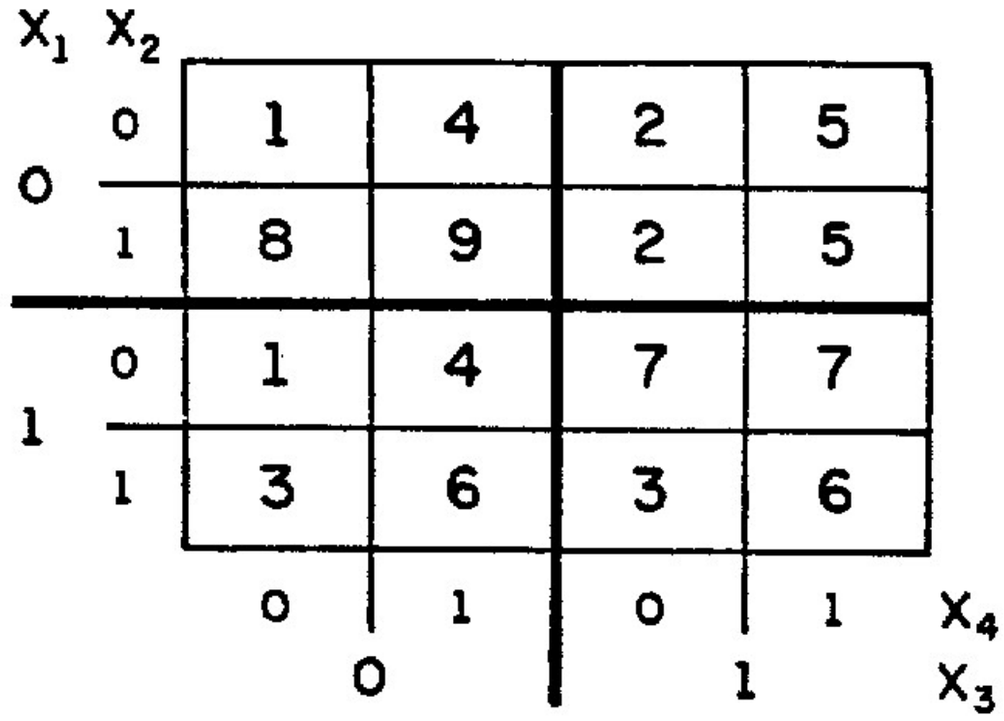
$\Delta_o(x_3) = 3$ $\qquad$ $\Delta_3(x_3) = 2$

$\Delta_o(x_4) = 1$ $\qquad$ $\Delta_4(x_4) = 1$

(The optimal cover shown in Fig. 16 was taken as the reference set for computing Δ_o).

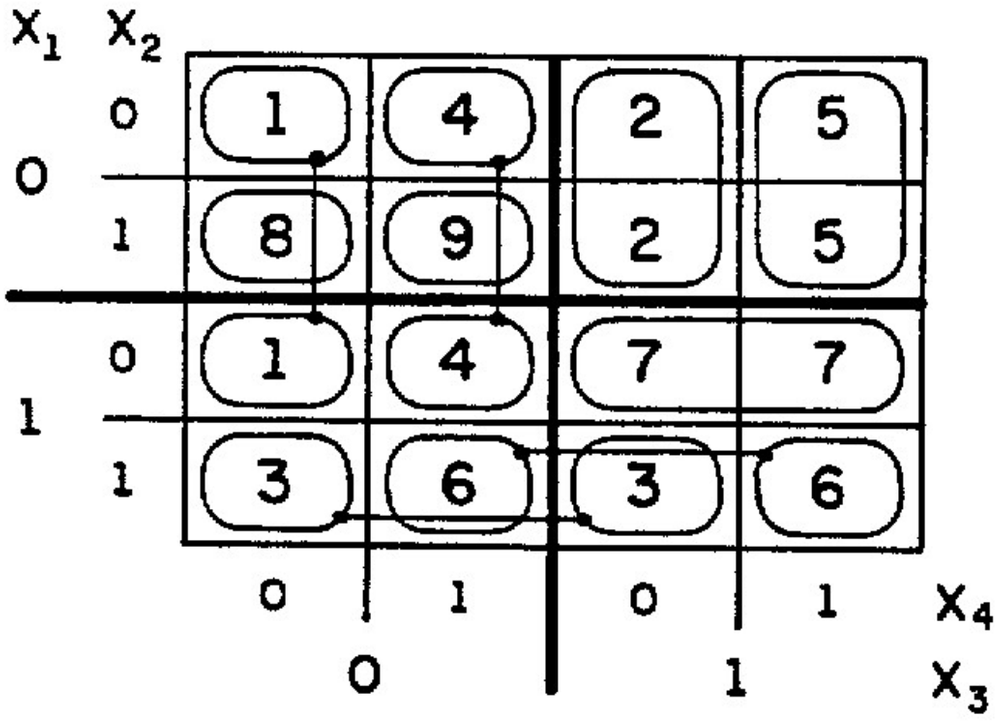
Both algorithms select test x_4 for the root, while this is the only test which does not produce an optimal tree (Fig. 17 and 18). It is easy to see that it is impossible to reject test x_4 by evaluating the effect of only one test on the decision diagram. In this case, one has to take into consideration the effect of a pair of tests, i.e., to apply a second degree algorithm.

Thus, to make algorithm S (or D) able to construct optimal decision tree in this case, one should compute, instead of Δ_o (Δ_1), the second degree cost



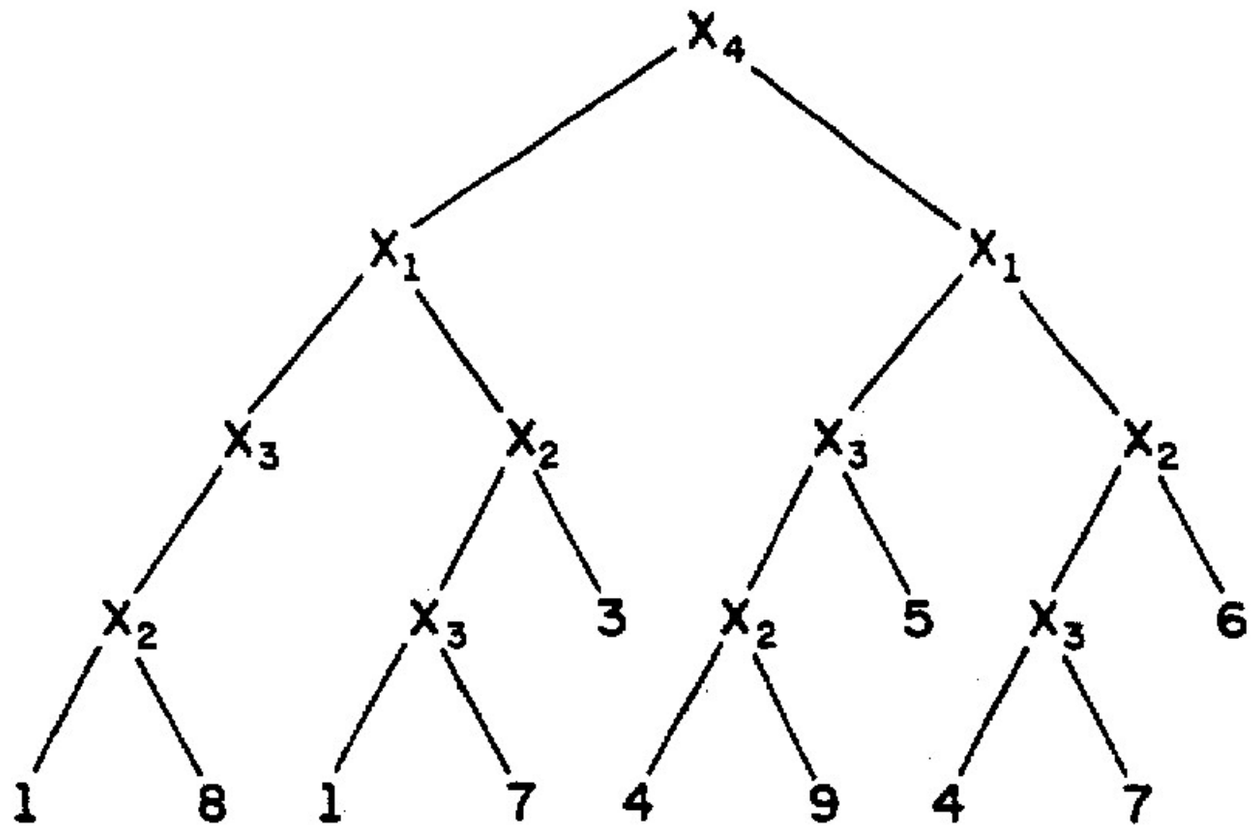
A counterexample to any
first degree algorithm.

Figure 15.



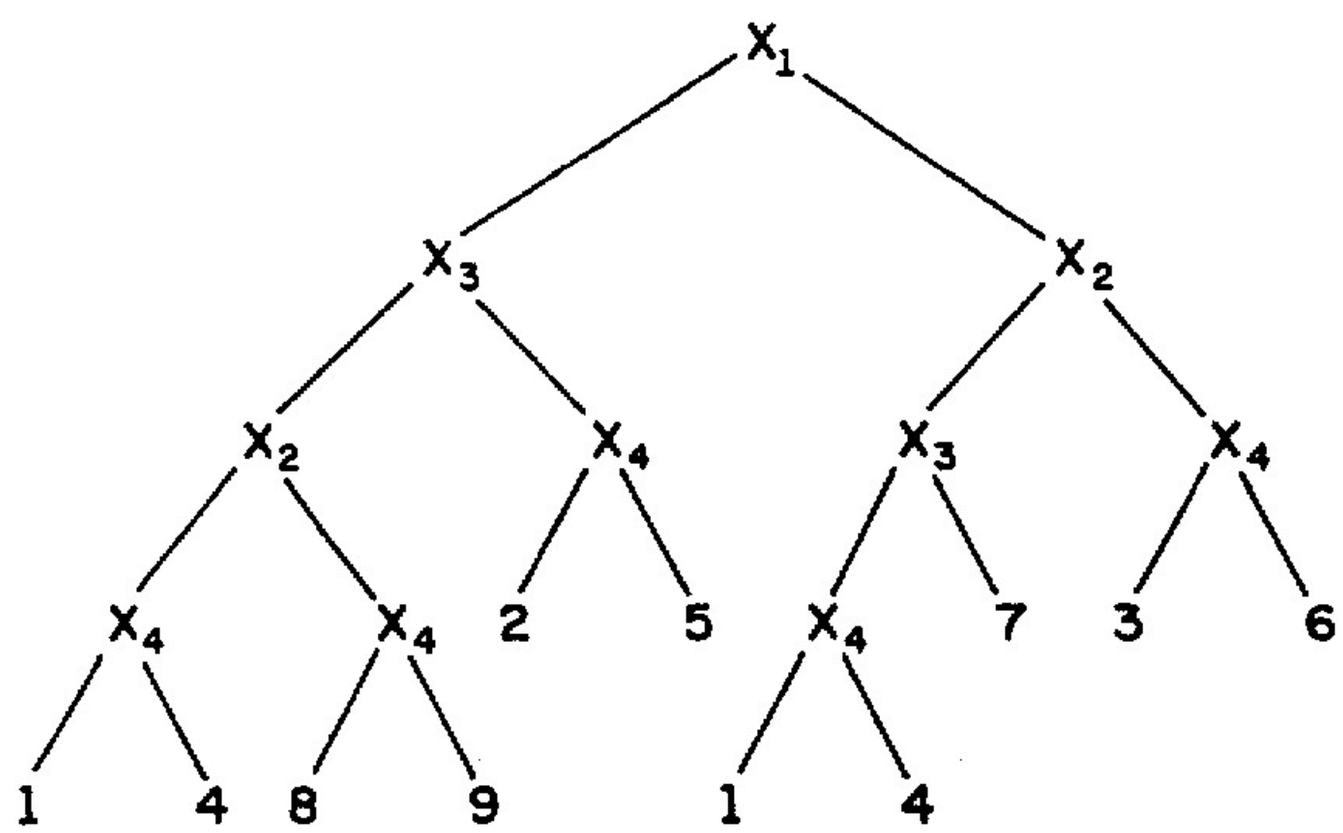
Optimal cover for decision diagram
in Figure 15.

Figure 16.



Sub-optimal decision tree for decision diagram in Fig. 15. (11 nodes)

Figure 17.



Optimal decision tree for decision diagram in Fig. 15. (10 nodes)

Figure 18.

estimate Δ_o^2 (or Δ_1^2). In Fig. 19, rectangular boxes include tests which can be selected for a given node of the tree together with the value of Δ_o. From this figure we see that:

$$\Delta_o^2(x_1) = \Delta_o(x_1) + \min \{\Delta_o(x_2/\bar{x}_1), \Delta_o(x_3/\bar{x}_1), \Delta_o(x_4/\bar{x}_1)\}$$
$$+ \min \{\Delta_o(x_2/x_1), \Delta_o(x_3/x_1), \Delta_o(x_4/x_1)\}$$
$$= 2 + 0 + 0 = 2$$
$$\Delta_o^2(x_4) = 1 + 1 + 1 = 3$$

where $\Delta_o(x_i/\bar{x}_j)$ and $\Delta_o(x_i/x_j)$ denote the estimates $\Delta_o(x_i)$ in the framework of subdiagrams $D(x_j=0)$ and $D(x_j=1)$, respectively.

Thus, $\Delta_o^2(x_1) < \Delta_o^2(x_4)$, and test x_4 will be rejected.

The author conjectures that for any algorithm of finite degree, there exists a decision diagram for which the algorithm will fail to produce the optimal tree.

The above example shows that by extending the order of algorithms, the class of conversion problems for which the algorithms produce the optimal tree is also extended.

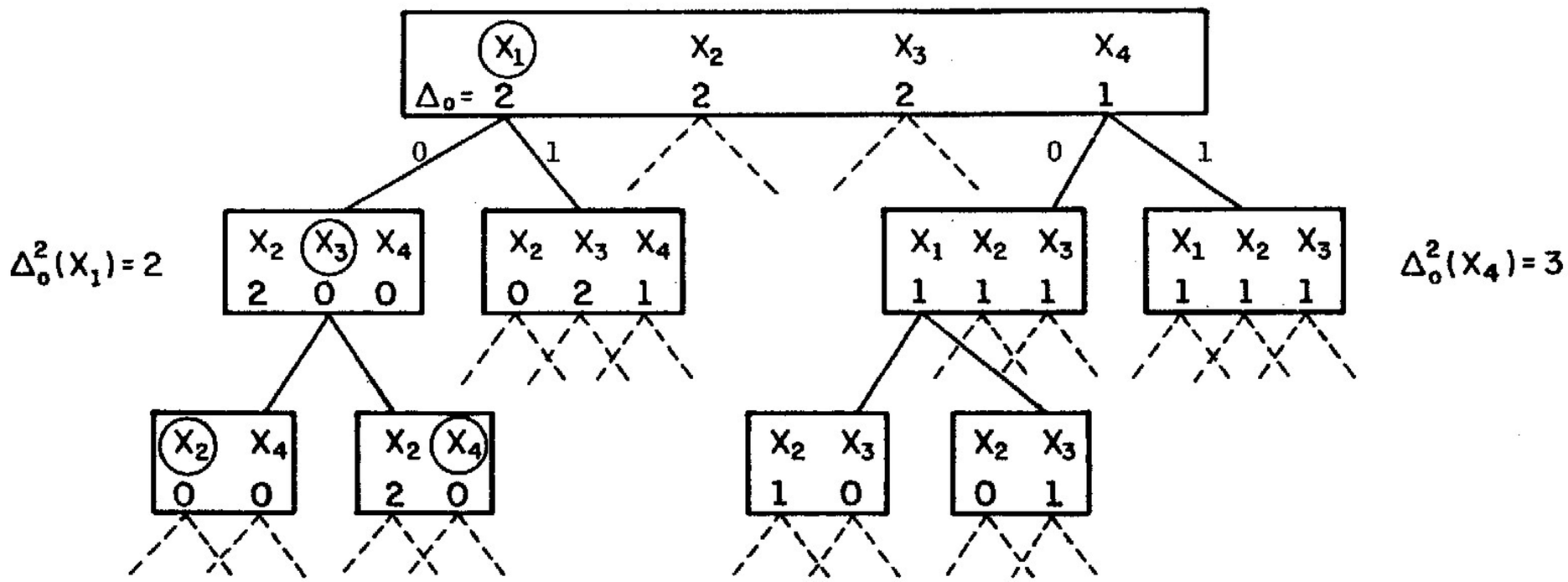
Let k_m be the maximum order of the estimate Δ_1 needed, that its computation for a test, a candidate for the root, reaches the leaves of the tree under construction. Obviously, $k \leq n$.

Theorem 3: If algorithm D employs the cost estimate Δ_1^k, then the resulting tree is guaranteed to be optimal.

Proof:

The estimate Δ_1 (unlike Δ_o) does not assume that any specific optimal (or irredundant) cover has to be first computed, and, therefore, is not effected by existence of more than one optimal (or irredundant cover). The value of Δ_1^k, for a test x_i is the minimum number of nodes, above the lower bound given by theorem 1, which can be in any tree whose root is x_i. The algorithm D selects the test for which Δ_1^k, is minimum. Therefore, the tree with so assigned root, and, recursively, other nodes, will have minimum number of nodes, i.e., will be optimal. ■

It is clear now, that by varying the degree k between 1 and k_m of the



○ denotes nodes in the optimal tree in fig. 17

Computation of the second degree cost estimates $\Delta_0^2(x_1)$ and $\Delta_0^2(x_4)$.

Figure 19.

estimate Δ_1^k, one obtains a spectrum of methods which differ in the trade-off between the computational efficiency and the degree of guarantee that the obtained tree is optimal.

3.5 Time optimal decision trees

In the case of converting a decision diagram to a tree corresponding to the time optimal program (called, for short, time optimal tree) one assumes that tests, x_i, are assigned costs, indicating the time needed for test evaluation, and that actions are assigned probabilities of their occurrence. The optimal tree is the tree which has the minimum weighted path length, i.e., the minimum value of

$$\sum_{j=1}^{\ell} p_j \cdot \text{path-cost}_j \qquad (11)$$

where

ℓ - the number of leaves

p_j - the probability of path j in the tree (suppose action A is assigned to path j, and the probability of action A is p_A. If the number of cells of action A in the decision diagram is c_a, and the number of cells in the complex corresponding to path j is c_j, then it is assumed that $p_j = \frac{c_j}{c_A} p_A$

path-cost_j - the sum of costs of the tests assigned to nodes on the path j.

Let L be a complex of decision class A. The complex L can be assigned the cost:

$$\text{cost}(L) = p_L \cdot \text{test-cost}(L) \qquad (12)$$

where $p_L = \frac{c_L}{c_A} p_A$

c_L - the number of cells in L

test-cost(L) - the sum of the costs of tests in complex L.

Selecting test x_i for a node of the tree corresponds to partitioning a diagram (subdiagram) D to d_i subdiagrams, $D(x_i=0)$, $D(x_i=1),\ldots,D(x_i=d_i-1)$. Let us assume initially, that R is an optimal cover of the diagram D, and R_j, $j=0,1,\ldots,d_i-1$, are the parts of the cover lying within the diagrams $D(x_i=j)$, $j=0,1,\ldots,d_i-1$, respectively.

Definition 10. The cost, cost(C), of a cover C is defined as the sum of the costs of its complexes.

The 'incremental' cost of selecting test x_i can be estimated as:

$$T_o(x_i) = \sum_j \text{cost}(R_j) - \text{cost}(R) \qquad (13)$$

Observe now, that since the costs of tests and probabilities P_A can have arbitrary values, the costs of different optimal covers can also be different (the situation is then different than it was in the case of space optimal trees). Consequently, in order to use $T_o(x_i)$ as a proper analogue of $\Delta_o(x_i)$ estimate, R in (13) should not be an optimal cover (def. 1), but a cost optimal cover, defined as a cover of the diagram of minimum cost.

In using the T_o estimate for test selection, it is computationally advantageous to ignore the component 'cost(R)' in (13), until a test is selected, and then to compute the 'complete' value of T_o (similarly as in computing Δ_1). The 'complete' value of T_o is needed, because the sum of T_o estimates, denoted ΣT_o, over all the nodes of the obtained tree, specifies (analogously to Σ_o) the maximum possible difference between the cost of the obtained tree and the optimal one.

In order to obtain an analogue T_1 to the Δ_1 estimate, both R and R_j in (13) should be the minimum cost covers of diagram D and diagrams $D(x_i=j)$, respectively. The sum of T_1 estimates, denoted ΣT_1, over the nodes of the obtained tree plays again the same role as Σ_1. The theorem 3 holds also for T_1^k.

It is interesting to observe that the dynamic programming algorithm by Schumacher and Sevcik [18] is equivalent, at the conceptual level, to computing T_1^n (i.e., the nth order estimate T_1). Some differences are that instead of the cost of a complex, they use an inversely related notion of the gain, defined as the difference between the sum of the costs of events in the complex and the cost of the complex. (The gain can equivalently be computed as the probability of the complex multiplied by the sum of the costs of tests which do not occur in the term expressing the complex.) Also, the order of computing T_1^n in [18]

is specified from the leaves of the tree up, while the definition of T_1^n suggests (but does not require) computing it from the root down. The way T_1^n is computed is, of course, a matter of implementation. The way it is done in [21] seems to be efficient, because it constructs the cost optimal cover (corresponding to the final tree) from the single cells up, step by step, building upon the intermediate results. This avoids a repetition of certain operations, which would occur, if one independently constructs covers of subsequent subdiagrams, going from the whole diagram to the individual cells.

It is easy to see, however, that Schumacher and Sevcik algorithm can be in certain cases very inefficient. This is because it always computes the most costly estimate T_1^n, even when a lower order estimate (much less costly) could produce the optimal decision tree (or 'sufficiently optimal', as measured by ΣT_o or ΣT_1).

The following example (taken from [18]) illustrates this observation.

Example 5

Fig. 20 presents a decision diagram and its cost optimal cover. The large size numbers in the cells indicate actions, and the small size numbers their probabilities. The action -1 indicates the logically excludable events (DON'T CARE-s), and the action 4 indicates ELSE events (assumed here as having 0 probability). The numbers in parentheses (at the axes) indicate costs of tests. We briefly illustrate here an application of algorithms S and D, in which first order estimates T_o and T_1 are used, instead of Δ_o and Δ_1.

1. Compute the cost, cost(R), of the optimal cover (see Fig. 20):

$$cost(R) = 0.2 \cdot 30 + 0.5 \cdot 30 + 0.3 \cdot 25 = 28.5$$

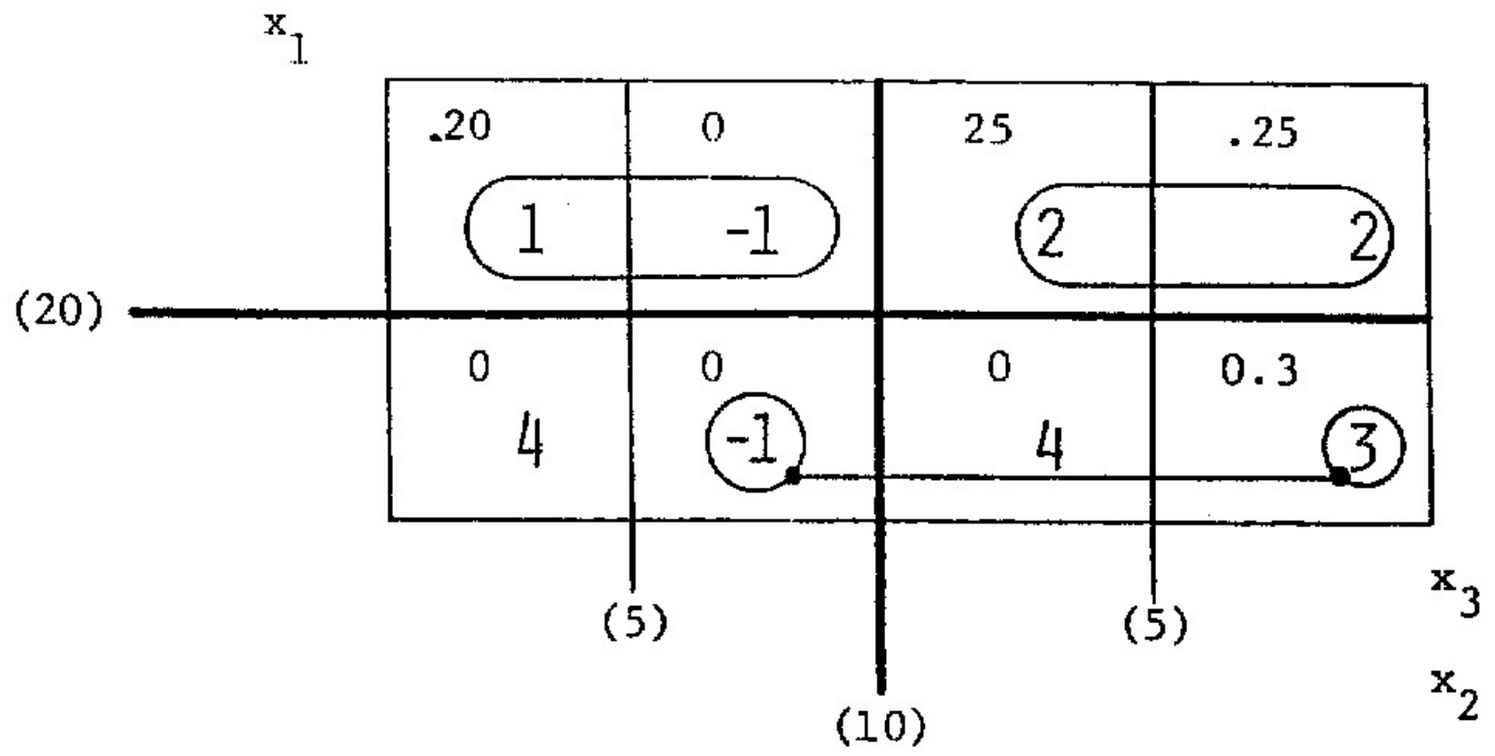
2. Compute T_o (and T_1) for each test:

$$T_o(x_1) = T_1(x_1) = \sum_{j=1}^{3} cost(R_j) - cost(R) = 0$$

$$T_o(x_2) = T_1(x_2) = 0.20 \cdot 30 + 0.5 \cdot 30 + 0.3 \cdot 35 - cost(R) = 3$$

$$T_o(x_3) = T_1(x_3) = 0.2 \cdot 35 + 0.25 \cdot 35 + 0.25 \cdot 25 + 0.3 \cdot 25 - cost(R) = 1$$

Test x_1 is selected for the root.



Decision diagram and the cost optimal cover for Example 5.

Fig. 20

3. Compute T_o (and T_1) for the test-candidates for the left descendant of the root.

$$T_o(x_2) = T_1(x_2) = 0$$

$$T_o(x_3) = T_1(x_3) = 1$$

Test x_o is selected.

4. Compute T_o (and T_1) for the test-candidates for the right descendant of the root.

$$T_o(x_2) = T_1(x_2) = 3$$

$$T_o(x_3) = T_1(x_3) = 0$$

The value $\Sigma T_o = \Sigma T_1 = 0$, thus, the tree is optimal. In fact, the tree is identical to the one obtained in [21], though its derivation required much less computation.

IV. SUMMARY

We have shown that the decision diagram introduced in the paper can be useful, both, as a conceptual model for describing algorithms, and as a practical tool for decision table design and conversion to space or time optimal decision trees. The advantage of the decision diagram is that rules in a decision table (or leaves of a tree) are represented as certain geometrical configurations, and relationships between the rules are represented as spatial relations between these configurations.

For this reason, the decision diagram can also be used as an educational aid, for visually illustrating concepts and algorithms related to decision tables and decision trees.

It may be of interest to the reader to mention here the results of an experiment done by the author in comparing the time spent in solving the same problem, using a conventional method and the decision diagram. The problem was to verify (check consistency, completeness and non-redundancy), reduce and convert to space optimal decision tree, the decision table shown in Fig. 1. The time spent on various phases of the problem by: A - the person who used a conventional method (a faculty member who teaches decision tables) and B - the author using the decision diagram, is given in Fig. 21. It should be mentioned that the decision tree obtained by person A had 1 more node than the optimal tree obtained using the decision diagram. Note, also, that the most of the time (10 minutes) in the decision diagram method was spent just on determining the decision diagram (which is rather a mechanical process, not requiring the knowledge of decision table algorithms).

	Time (min)	
	Using a conventional method	Using decision diagram
Draw diagram	-	1
Draw complexes in the diagram	-	9
Reduce table (determine cover)	13	1
Verify	2	0' 5"
Convert to tree	2' 30"	2
TOTAL	17' 30"	13'5"

Time spent on various phases of the problem using a conventional method and the decision diagram.

Figure 21

The concept of kth degree conversion algorithm , also introduced in the paper, permits one to generate a spectrum of conversion algorithms, differing in the trade-off between the computational efficiency and the degree of guarantee of the decision tree optimality. The algorithms S and D were shown to be applicable for both space and time optimal conversion, and they can use cost estimates of a different degree. When algorithms do not produce the optimal tree, they gave a measure of the maximum possible difference between the obtained and the optimal trees.

ACKNOWLEDGMENTS

The research reported in this paper was supported by a grant from the National Science Foundation, NSF MCS 76-22940.

The author thanks Professor Gary Kampen for stimulating discussions and comments, and Tom Dietterich for proofreading of the paper.

REFERENCES

1. Alster, T. M. Heuristic algorithms for constructing near-optimal decision trees. Report No. UIUCDCS-R-71-474, Department of Computer Science, University of Illinois, Urbana, IL., Aug. 1971.

2. Bayes, A. T. A dynamic programming algorithm to optimise decision table code, Australian Computer T. 4 (May 1973), 77-79.

3. Fisher, D. L. Data documentation and decision tables. Comm. ACM, 18 (Jan. 1965), 26-31.

4. Ganapathy, S., Rajaraman, V. Information theory applied to the conversion of decision tables to computer programs. Comm. ACM 16, 9 (Sept. 1973), 532-39.

5. Jarvis, J. M. An analysis of programming via decision table compilers. SIGPLAN Notices (ACM Newsletter) 6,8 (Sept. 1971), 30-32.

6. King, P. J. H. Conversion of decision tables to computer programs by the rule mask technique. Comm. ACM 9, 11 (Nov. 1966), 796-801.

7. Kirk, G. W. Use of decision tables in computer programming. Comm. ACM 8, 1 (Jan. 1965), 41-43.

8. Larson, J., Michalski, R. S. AQVAL/1 (AQ7) User's guide and program description. Report No. UIUCDCS-R-75-731, Department of Computer Science, University of Illinois, Urbana, IL. June 1971.

9. Michalski, R. S. On the quasi-minimal solution of the general covering problem. Proceedings of the V international symposium on information processing (FCIP 69), Vol. A3 (Switching Circuits), Yugoslavia, Bled, Oct. 8-11, 1969, 125-127.

10. Michalski, R. S. A geometrical model for the synthesis of interval covers. Report No. UIUCDCS-R-71-731, Department of Computer Science, University of Illinois, Urbana, IL., June 1975.

11. Michalski, R. S. Synthesis of optimal and quasi-optimal variable-valued logic formulas. Proceedings of the 5th International Symposium on Multiple-Valued Logic, Bloomington, Indiana, May 13-16, 1975, 76-87.

12. Michalski, R. S. A Planar Geometrical Model for Representing Multi-dimensional Discrete Spaces and Multiple-Valued Logic Functions. Report No. UIUCDCS-R-78-897, Department of Computer Science, University of Illinois, Urbana, IL., January 1978.

13. Michie, D. ALl - a package for generating strategies from tables. SIGART Newsletter, No. 59, 1976.

14. Pollack, S. Conversion of limited-entry decision tables to computer programs. Comm. ACM, 8, 11 (Nov. 1965), 677-82.

15. Pooch, U. W. Translation of decision tables. Computing Surveys 6, (June 1974), 125-51.

16. Rabin, T. Conversion of limited-entry decision tables into optimal decision trees: fundamental concepts. SIGPLAN Notices (ACM Newsletter) 6, (Sept. 1971), 68-71.

17. Reinwald, L. T., and Soland, R. M. Conversion of limited-entry decision tables to optimal computer programs - I: Minimum average processing time. J. ACM 13, 3 (July 1966), 339-58., II: Minimum storage requirement. J. ACM 14, 4 (Oct. 1967), 742-55.

18. Schumacher, H., Sevcik, K. C. The synthetic approach to decision table conversion. Comm. ACM, 19, 6 (June 1976), 343-51.

19. Shwayder, K. Extending the information theory approach to converting limited-entry decision tables to computer programs. Comm. ACM 17, 9 (Sept. 1974), 532-37.

20. Shwayder, K. Conversion of limited-entry decision tables to computer programs - a proposed modification to Pollack's algorithm. Comm. ACM 14, 2 (Feb. 1971), 69-73.

21. Verhelst, M. The conversion of limited-entry decision tables to optimal and near-optimal flowcharts: two new algorithms. Comm. ACM 15, 11 (Nov. 1972), 974-80.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-78-898	2.	3. Recipient's Accession No.

4. Title and Subtitle

DESIGNING EXTENDED ENTRY DECISION TABLES AND OPTIMAL DECISION TREES USING DECISION DIAGRAMS

5. Report Date March 1978

6.

7. Author(s) Ryszard S. Michalski

8. Performing Organization Rept. No. UIUCDCS-R-78-898

9. Performing Organization Name and Address

University of Illinois at Urbana-Champaign
Department of Computer Science
Urbana, Illinois 61801

10. Project/Task/Work Unit No.

11. Contract/Grant No.

12. Sponsoring Organization Name and Address

13. Type of Report & Period Covered

14.

15. Supplementary Notes

16. Abstracts The paper introduces the concept of a decision diagram and shows its application to designing extended entry decision tables and converting them to space or time optimal decision trees. A decision diagram is a geometrical representation of a decision table by means of a planar model of a multidimensional discrete space as described in [12].

Two algorithms for optimal (or suboptimal) space or time conversion are described using decision diagrams. These algorithms are basically decomposition algorithms, but by varying their degree (def. 5), one can obtain a spectrum of algorithms, differing in the trade-off between the computational efficiency and the degree of guarantee that the solution is optimal. When the algorithms do not guarantee the optimality, they give a measure of the maximum possible distance between the obtained and the optimal trees.

17. Key Words and Document Analysis. 17a. Descriptors

Limited Entry Decision Tables, Extended Entry Decision Tables, Decision Trees, Conversion Algorithms, Decision Diagram, Logic Diagram.

CR Categories: 8.3

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
Release Unlimited	UNCLASSIFIED	57
	20. Security Class (This Page) UNCLASSIFIED	22. Price