Copyright © 1979, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

APPROXIMATE SOLUTION FOR THE CONVERSION OF DECISION TABLES PROBLEM

by

Malik Ghallab

Memorandum No. UCB/ERL M79/60

24 September 1979

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

APPROXIMATE SOLUTION FOR THE CONVERSION OF DECISION TABLES PROBLEM

Malik Ghallab

Department of Electrical Engineering and Computer Sciences
and the Electronics Research Laboratory
University of California, Berkeley, California 94720

ABSTRACT

The problem of conversion of Decision Tables into optimal Decision Trees is studied. Its complexity is characterized as NP-Hard in the strong sense. An approximation algorithm is developed and analyzed. Some running experiments on random data are described and results illustrating the average behavior of the proposed algorithm are given.

Research sponsored by the National Science Foundation Grant INT78-09263.

1. Introduction

The main purpose of this paper is to characterize the complexity of a combinatorial optimization problem, both from the exact optimization point-of-view and from the approximation one. The worst case complexity is characterized on a theoretical basis, but for the much more difficult average complexity, we retreated to an empirical characterization.

The analysis of the complexity of a hard combinatorial problem is not the only goal pursued here. We are also concerned about practical ways for solving our problem in a satisfactory manner.

The paper is focused on the Decision Tables Conversion problem. Decision Tables have been known since the beginning of the 60's and continue to be widely studied. A 1974 survey paper [34] listed more than 100 references dealing with Decision Tables. A complete bibliography of the subject would be much larger today.

Decision Talbes have been essencially used in information processing as a programming tool powerful in simplifying flowcharts, and in the documentation, verification and design of programs. Programmers interested in using Decision Tables can find at least 40 to 50 software packages available today on the market, ranging from processors for almost any programming language (including LISP [42]), to complete compilers [27].

Applications of Decision Tables are not limited to computer programming. They range over a large class of problems, including information retrieval and file organization, simulation, testing and trouble shooting, medical diagnosis and Pattern Recognition [1, 9, 43].

Although the Decision Table Conversion problem is in itself interesting, we feel that the approach described here is illustrative and typical for a broad class of combinatorial problems.

The paper is divided into 5 sections. The next section reviews Decision Tables definitions and properties, and states the Conversion problem. Section 3 is devoted to the characterization of the worst case complexity. Section 4 develops a particular algorithm and gives the results of running experiments. The last section presents some concluding remarks together with interesting open problems related to our subject.

We have paid close attention to the details and clarity of the presentation. Whenever possible, formal expressions and heavy notations were avoided. Numerous examples are developed throughout the text.

2. Definitions and Properties of Decision Tables

This section restates briefly the basic definitions and properties of Decision Tables. It can be skipped by a reader familiar with Decision Table literature [10, 23, 33, 34].

In the following, a condition x_i will refer to a scalar function which maps some input data into a discrete range R_i. An evaluation cost P_i of x_i is involved as a constant charge each time x_i is tested.

Given a set of N conditions $\{x_1,\ldots,x_N\}$, a simple event is an element of the cartesian product $R_1 \times \ldots \times R_N$. If I is any subset of $\{1,\ldots,N\}$, a composite event is an element of the product: $\prod_{i \in I} R_i$.

In a simple event all the N conditions have a specified value. But in a composite event, some conditions have a "don't-care" value. A composite event is equivalent to a set of simple events. This set is defined by expanding the composite event, i.e. by assigning to its don't-care conditions all the combinations of their values. An example will help clarify our notation: Let $\{x_1, x_2, x_3, x_4\}$ be 4 binary conditions, i.e. $R_i = \{0,1\}$ for $i=1,2,3,4$; $(1,0,\phi,\phi)$ is a composite event, x_3 and x_4 are its don't-care conditions; its expansion is: $(1,0,\phi,\phi) = \{(1,0,0,0); (1,0,0,1); (1,0,1,0); (1,0,1,1)\}$. Two events like (1,0,1,0) and (1,0,1,1) are <u>adjacent</u>. They differ only by one condition, x_4, which is their <u>consensus variable</u>. To compress two adjacent events is to assign a don't-care value to their consensus variable: (1,0,1,0) and (1,0,1,1) are compressed into $(1,0,1,\phi)$. Since events are sets (of simple events), we employ the usual terminology: disjoint events, intersecting or overlapping events,...etc. Finally we assume that some statistics are known which enable us to determine the probability of occurance of any event.

The N conditions of some given set $\{x_1,\ldots,x_N\}$ are not necessarily independent. A general formulation of <u>dependency relations</u> consists of a set D of impossible events, i.e. events which state the forbidden combinations of conditions values. An impossible event has of course a null probability of occurance. (For a discussion of dependency relations in Decision Tables see [12, 17, 18].) We assume nevertheless that the n conditions can be evaluated in any order.

A <u>decision rule</u> is a relation which specifies the action to be taken when some particular event occurs. A <u>decision table</u> is a set of decision rules.

More formally, we define a decision table as a quintuple

T = (X, A, D, E, F) where:

$X = \{x_1,\ldots,x_N\}$ is a set of N conditions;

$A = \{a_1,\ldots,a_M\}$ is a set of M labels called actions;

D is a set of impossible events, or dependency relations on X;

E is a set of decision events for which an action is specified in the table;

F is a function mapping E into A.

Furthermore, for a table T the following data are assumed to be known:

- The evaluation costs of the N conditions: $(\ell_1,\ldots,\ell_N)$;
- The statistics on the decision events, usually given by a probability distribution over E and by the assumption that inside a composite event the probabilities of occurrence are equidistributed.

Figure 1 illustrates the graphical representation of a decision table:

	.1	.2	.1	.2	.3	.1			ℓi
x_1	1	0	ϕ	0	ϕ	0	1	1	10.
x_2	1	1	1	ϕ	0	0	0	ϕ	50.
x_3	0	ϕ	1	0	0	1	ϕ	1	30.
x_4	ϕ	1	0	0	1	ϕ	0	1	20.
		a_4			a_2		D		

Figure 1

This table contains 4 binary conditions and 2 actions. Each internal column is an event. The last row gives the action to which an event is mapped, or shows D for dependency events. The condition's costs appear in the last column, and the event's probability distribution is given in the first row. Notice that, since in this example the events are non-overlapping, the probability distribution sums to 1.

There are many different forms of decision tables described in the literature. An expanded decision table is one where the elements of D and E are restricted to be simple events. Otherwise a table which contains don't-cares is a <u>compressed</u> decision table. An expanded table can be compressed by the consensus variable technique applied to events mapped to the same action or belonging to D.

A set X containing only binary conditions, as in the example of fig. 1, leads to a limited-entry decision table. An extended-entry decision table provides provision for multivalued conditions.

A decision table is consistant if any 2 events in E mapped to 2 distinct actions are disjoint. A table is complete if any simple event of the cartesian product $R_1 \times \ldots \times R_N$ is included either into a decision event or into an impossible event of D. An incomplete table can be completed implicitly by an Else-rule, i.e. a rule which states a particular action for all the unspecified events.

In a well studied particular case, each condition defines a partition over the set of actions. This subclass of Decision Tables is refered to in the literature as Identification Procedures [5, 6] or "Questionnaires" [3, 30, 31]. Among the particulars of Identification Procedures, the set E contains exactly M simple events, one for each action; and all the remaining events are dependency relations.

Two steps are involved in the processing of a decision table. In the first step the consistency and eventually the completeness of the table are checked. (See [8, 17] about this step.)

The second step is concerned with the conversion of the decision table into a computer program. As discussed in the literature [2, 16, 33, 46-48], the most general method uses the Decision Tree approach.

A decision tree t on N conditions $\{x_1,\ldots,x_N\}$ and M actions $\{a_1,\ldots,a_M\}$ is a tree structure where:

i) each node is labeled by some condition x_i;

ii) the branches departing from a node x_i correspond to the elements of R_i, range of x_i;

iii) each leaf or terminal node is labeled by some action a_k; and

iv) a path from the root to a leaf crosses each condition at most once.

It is easy to see that each leaf of a decision tree corresponds to an event (a simple event if the path leading to this leaf crosses all the conditions). Two distinct leaves correspond to two disjoint events, and any simple event of $R_1 \times \ldots \times R_N$ is covered by some leaf. Thus a decision tree defines a partition on $R_1 \times \ldots \times R_N$ and maps the element of this partition into $\{a_1, \ldots, a_M\}$. By definition a decision tree is complete and consistant.

Given a complete and consistant decision table T = (X, A, D, E, F), and a tree t on X and A, t <u>translates</u> T if any simple event is either included in D or mapped by t and T into two identical actions.

The following tree (fig. 2) translates the table of fig. 1.

```
( 2 )→→→→( 1 )→→→→¤:1
  ↓        ↓
  ↓        ↓
  ↓      ( 4 )→→→→¤:1
  ↓        ↓
  ↓        ↓
  ↓      ( 3 )→→→→¤:1
  ↓        ↓
  ↓        ↓
  ↓        ¤:2
  ↓
  ↓
  ↓
  ¤:2
```

Figure 2

<u>Note</u>: By convention, horizontal branches correspond to the value 1 of the binary condition, and vertical branches to the values 0.

The cost of a leaf of a decision tree is the sum of the evaluation costs of all the conditions crossed in the path from the root to this leaf. The weight of a leaf is the probability of appearance of the event associated to this leaf. The mean decision cost of a tree t, noted $\Psi(t)$, is the weighted sum of the cost of its leaves.

Let us compute for example the mean decision cost of the tree of fig. 2, with the data of table 1, taking the leaves in depth first left to right order:

leaf 1: cost: $\ell_2 = 50.$

weight: $Pr[(\phi,0,\phi,\phi)] = \frac{1}{2} Pr[(0,\phi,0,0)]+Pr[(\phi,0,0,1)]+Pr[(0,0,1,\phi)] = .5$

leaf 2: cost: $\ell_2+\ell_1+\ell_4+\ell_3 = 110.$

weight: $Pr[(0,1,0,0)] = \frac{1}{2} Pr[(0,\phi,0,0)] = .1$

leaf 3: cost: $\ell_2+\ell_1+\ell_4+\ell_3 = 110.$

weight: $Pr[(0,1,1,0)] = \frac{1}{2} Pr[(\phi,1,1,0)] = .05$

leaf 4: cost: $\ell_2+\ell_1+\ell_4 = 80.$

weight: $Pr[(0,1,\phi,1)] = .2$

leaf 5: cost: $\ell_2+\ell_1 = 60.$

weight: $Pr[(1,1,\phi,\phi)] = Pr[(1,1,0,\phi)]+\frac{1}{2}Pr[(\phi,1,1,0)] = .15$

Finally the cost of t is

$$\Psi(t) = 50\times.5 + 110\times.1 + 110\times.05 + 80\times.2 + 60\times.15 = 66.5$$

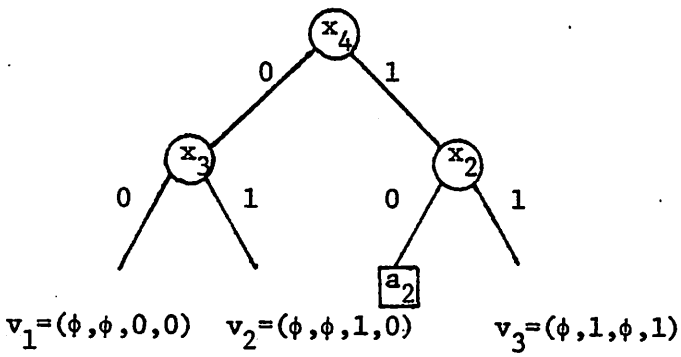
There are $\prod_{k=o}^{N-1} (N-k)^{r^k}$ possible trees translating a table of N r-ary conditions (i.e. for $1\leq i\leq N; |R_i| = r$) [8,41].

The Decision Tables Conversion (DTC) problem is to find a tree translating a table, minimal for the mean decision cost criterium.

We end this section by defining partial trees, subtables and an algorithm which converts decision tables into decision trees.

A <u>partial decision tree</u> τ is either the empty tree τ_o or a tree where at least one path from the root does not end in a leaf, i.e. in a node labeled by an action. Such a path is called an <u>open branch</u>. As to any other path in a decision tree, an event is associated to an open branch relative to some decision table T. The don't-care conditions of this event, the element of E and D which intersect with it and their corresponding actions define a subtable associated to an open branch. The empty partial tree τ_o has only one open branch with the entire initial table associated to it.

For the table of figure 1, figure 3 illustrates a partial tree with 3 open branches, and shows the subtable associated to the branch $v_1 = (\phi,\phi,0,0)$



	.05	.2		ℓi
x_1	1	0	1	10.
x_2	1	ϕ	0	50.
	a_1	a_2	D	

Figure 3

The following algorithm is the main frame of almost all the heuristics published for the Decision Tables Conversion (DTC) problem.

Algorithm A1.

Input: a decision table T;

1. Start: with an empty partial tree $\tau \leftarrow \tau_o$

2. Do While τ is a partial tree;

2.1 Take in τ an open branch v;

2.2 If subtable corresponding to v contains only one action a_k Then Do;

2.2.1 Expand τ by completing the open brach v with a leaf containg a_k;

2.2.2 End;

2.3 Else Do;

2.3.1 Choose in the subtable corresponding to v a condition x_j;

2.3.2 Expand τ by appending a node labeled x_j to the branch v;

2.3.3 End;

2.4 End;

Output: a decision tree $t \leftarrow \tau$ translating T.

We remark that:

i) 2.3.1 is a nondeterministic step

ii) if ℓ is the total number of leaves of the tree generated, algorithm A1 has a computing time in $O(\ell)$.

Although all the results of section 3 and the algorithms of section 4 remain valid for extended-entries decision tables, to simplify the presentation we will restrain ourselves in the remaining to decision tables with binary conditions.

3. Complexity of the Decision Tables Conversion (DTC) Problem

Although the DTC problem has been widely studied, no polynomial-time algorithm is known neither for solving exactly the problem, nor for giving a guaranteed or even a "good" approximation. In this section the worst case complexity of the DTC problem is characterized and some of its particular aspects are discussed.

We use in the following the polynomial reduction approach defined in [14]. Let us recall some definitions from [7] adapted to our case:

• For $\varepsilon>0$, t is an <u>ε-optimal</u> tree translating a table T if:

$[\Psi(t)-\Psi(t^*)]/\Psi(t^*) \le \varepsilon$, where t^* is an optimal tree translating T.

• An algorithm $A(\varepsilon)$ is an <u>approximation scheme</u> for the DTC problem if for any table T and $\varepsilon>0$, $A(\varepsilon)$ generates an ε-optimal tree translating T.

• $A(\varepsilon)$ is a <u>polynomial-time approximation scheme</u> if for any table T and $\varepsilon>0$, the running time of $A(\varepsilon)$ is bounded by a polynomial in the size of T.

• $A(\varepsilon)$ is a <u>fully polynomial-time approximation scheme</u> if its running time is bounded by a polynomial in the two variables: $(\frac{1}{\varepsilon})$ and the size of the table.

Let us now define a function S, size of a decision table, such that any T=(X, A, D, E, F) can be coded into a string whose length is polynomial

in S[T]. For this definition, a first remark is that E and D are not uniquely defined. A composite event can be expanded, or conversely we can compress a set of events mapped to the same action. Eventually $e=|E|$ and $d=|D|$ can be minimized with the algorithm of [26] or an equivalent algorithm. But this minimization is itself an NP-Hard problem [40], and furthermore unnecessary for the conversion of Decision Tables. Nevertheless we will assume that any table T is given in a form where e and d are polynomially related to e_{min} and d_{min} respectively.

A second remark is that $N=|x|$ and $M=|A|$ are not sufficient to characterize the size of a table since the number of events can be in $O(2^N)$ with no possible reduction even for $M<<N$. (For example M=E, to A_1 we map the 2^{N-1} simple events with an odd number of "1", and to a_2 the 2^{N-1} simple events with an even number of "1").

Finally a proper definition would be $S[T]=\max[N, e, d]$ since $M\le e$. For practical purpose, we will take $S[T]=N$ each time e and d are polynomially related to N.

In order to characterize the complexity of the DTC problem we will use a reduction of the <u>Node Cover</u> (NC) problem. Given a graph $G=(X,A)$, the NC problem consists in finding a minimum number of nodes of X which cover all the edges of A. NC is an NP-complete problem [14].

<u>Theorem</u>: The Decision Tables Conversion problem is NP-Hard in the strong sense.

<u>Proof</u>: Let us define a transformation which maps any instance of the NC problem into a particular instance of the DTC problem. From $G=(X,A)$, a graph with N nodes $X=\{x_1,\ldots x_N\}$, and M edges $A=\{a_1,\ldots,a_M\}$, we built the limited-entry decision table $T=(X, A', E, D, f)$ where

(i) $A' = A \cup \{a_{M+1}\}$

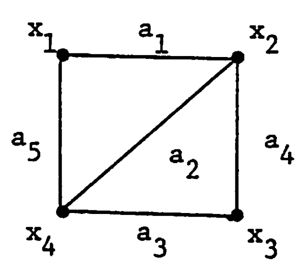
(ii) E has M+1 simple events, one per action:

- for $1 \leq k \leq M$, if a_k is in G the edge (x_i, x_j), then in the event of E mapped by f to the action a_k, x_i and x_j have value 1 and the other conditions have value 0;
- in the event mapped to a_{M+1}, all the conditions have value 0.

(iii) D contains all the other events. (Note: D may be explicited in less than N^3 events: $\binom{3}{N}$ composite events which have 3 conditions with value 1 and (N-3) don't cares, and $\binom{2}{N} - M$ remaining simple events which have 2 conditions with value 1 and (N-2) with value 0.)

(iv) The N conditions in T have $\rho_j = 1$ as evaluation cost; and for $1 \leq k \leq M$ the event mapped to a_k has probability zero, the event mapped to a_{M+1} having probability 1.

The example of figure 4 illustrates this transformation from G to T.

G: 

T:

	0	0	0	0	0	1						ρ_j
x_1	1	0	0	0	1	0	1	1	1	ϕ	1	1
x_2	1	1	0	1	0	0	1	1	ϕ	1	0	1
x_3	0	0	1	1	0	0	1	ϕ	1	1	1	1
x_4	0	1	1	0	1	0	ϕ	1	1	1	0	1
	a_1	a_2	a_3	a_4	a_5	a_6	D					

Figure 4

Since only one simple event is mapped to each action in T, any tree translating T has exactly M+1 leaves, one per action. Clearly, an optimal tree translating T is one which has a minimum number of nodes in the path leading to the leaf a_{M+1}, since this is the only leaf with a non null weight.

Let $x_{i_1},\ldots,x_{i_k}$ be the conditions crossed in the path leading to a_{M+1}: The event associated to this leaf a_{M+1} (has $x_{i_1} = x_{i_2} = \cdots = x_{i_k} = 0$ and the other conditions are don't-cases) intersects with only action a_{M+1}. It follows that the complementary set of events ($x_{i_1} = 1$ <u>or</u> $x_{i_2} = 1$ <u>or</u> $\cdots$ <u>or</u> $x_{i_k} = 1$) overlaps with the other actions $a_1,\ldots,a_M$. Thus, in graph G the nodes $x_{i_1},\ldots,x_{i_k}$ cover all the edges $a_1,\ldots,a_M$.

It is then obvious that any optimal tree translating T defines a solution to the NC problem in G and conversely.

Finally to end our proof we notice that:

(i) T is generated from G by a <u>pseudo-polynomial transformation</u> (T is defined in $O(N^4)$ steps: N^3 events each being an N-tuple); and

(ii) Node Cover is not a <u>number problem</u> (the magnitude of the largest number intervening in NC is bounded by N). □

From the precedent theorem, and from Theorem 1 of [7] one easily deduces that the DTC problem cannot be solved by a fully polynomial approximation scheme unless P = NP.

Another immediate result follows from the precedent theorem:

<u>Corollary</u>: For any ε such that $0 < \varepsilon \leq 1/N$, the DTC problem on decision tables of N conditions or less is an NP-Hard ε-approximation problem.

<u>Proof</u>: We use the precedent reduction of the NC problem to the DTC problem. If t is an ε-optimal tree of cost k, any other tree t' strictly better than t will cost at most k-1. The ratio:

$$[\psi(t)-\psi(t')]/\psi(t') \geq 1/(k-1) > 1/N \geq \varepsilon \ .$$

For $\varepsilon \leq 1/N$, t is ε-optimal if and only if t is optimal. The rest of the argument applies as before. □

The non-constant bound for ε weakens this corollary. Nevertheless, it is important to know that in the worst case, there are decision tables and values of ε for which an ε-optimal solution is as hard to find as an optimal one. We can rule out the possibility of an approximation scheme in which computing time is not only in $O(N^k/\varepsilon^r)$ but also in $O(N^{k/\varepsilon})$.

Until now we did not consider a basic question about the complexity of our problem: is it (more rigorously is the corresponding "decision" problem) in the class NP? Surprisingly it is not. A nondeterministic algorithm must be able to check, for any tree t translating a table, if the cost of t is less than some constant or not. The cost of t is computed in a time proportional to the number of its leaves. We must then either prove that the number of leaves is bounded by a polynomial of the size of the table, or exhibit a counterexample.

Consider the following table (figure 5):

x_1	0	1	1	$\cdots$	1	1	1
x_2	ϕ	0	1	$\cdots$	1	1	1
x_3	ϕ	ϕ	0	$\cdots$	1	1	1
$\vdots$				$\ddots$	$\vdots$	$\vdots$	$\vdots$
x_{N-1}	ϕ	ϕ	ϕ	$\cdots$	0	1	1
x_N	ϕ	ϕ	ϕ	$\cdots$	ϕ	0	1
	a_1	a_2	a_3		a_{N-1}	a_N	a_{N+1}

Figure 5

This table T has N independent conditions ($D = \emptyset$) and N+1 events and actions. Let t be the tree translating T which tests x_N at the root, x_{N-1} at the 2 nodes of level 1, ..., x_{N-k} at the 2^k nodes of level k, and so on until all the paths can reach a leaf. Any event where x_1 is a don't care condition overlaps necessarily with more than one action. No action can then be reached in t until x_1 is tested in level N. It follows that t has 2^N leaves. (Notice that the number of leaves cannot be reduced since no internal node in t is redundant, i.e. has its two subtrees identical).

Thus the DTC problem is not in the class NP. But for the particular case of Identification Procedures, since only one simple event is mapped to each action, any decision tree has exactly M leaves, one per action, and the problem of Conversion of Identification Procedures into optimal trees is NP-complete. (See [11] for a direct proof which uses a reduction of the exact cover problem with 3-elements subsets.)

No interesting subclass of the DTC problem is known to have a polynomial time algorithm. An almost obvious case is the identification procedure which has M = N+1 actions; for $1 \leq k \leq N$, action a_k corresponds to the simple event where $x_k = 1$ and $x_j = 0$ for $j \neq k$; and the simple event mapped to a_{N+1} has $x_1 = \cdots = x_N = 0$. The DTC problem on this identification procedure is exactly the well known problem of sequencing N jobs in one machine with processing times, delay rates and no precedence constraints. It can be solved in $O(N \log N)$ [25].

To end this section let us analyze briefly the complexity of Decision Tables consistency and completeness problems. The consistency of a table can be checked by verifying that two events of E mapped to two distinct actions are disjoints. This is done in at most $O(Ne^2)$

comparisons. (Recall $e = |E|$.)

The completeness problem is much harder. In fact, an event is a conjunctive clause, and a decision table can be regarded as a disjunction of clauses like a formula of propositional calculus. The completeness problem is then exactly the tautology problem known to be NP-complete [39].

In a particular case where all the events of a table are mutually disjoint (even if mapped to the same action or belonging to D), the completeness can be checked in $O(N \times (e+d))$. It is sufficient to compute the sum $\sum 2^{\alpha_i}$ over all the events of $E \cup D$, α_i being the number of don't-cares of the considered event. The table is complete if $\sum 2^{\alpha_i} = 2^N$. But even in this particular case, the definition of an Else-rule, if needed, remains an NP-complete problem.

4. A Practical Approximation Scheme for the DTC Problem

As one may expect from the previous section, the exact algorithms known for the DCT problem, based on Dynamic Programming [22,41] or on Branch and Bound [8,24,36,37] are exponential. The usual move in similar problems is to retreat from exact algorithms to heuristic algorithms. Many heuristics have been proposed for the DCT problem [28,32,35,44,45, 47,48], but to the author's knowledge, no one has been proved to be polynomial.

Heuristic algorithms are generally based on algorithm A1 of section 2, to which they add some particular rule in order to choose a "good" condition in step 2.3.1. Their complexity is directly related to the number of leaves of the tree generated in the worst case. Other heuristics, based on the Dynamic Programming approach are systematically exponential.

For example, [41] shows that the heuristic of [4] converts any table in $O(N^2 \times 2^N)$ steps.

We conjecture that only some "poor" heuristics, which do not take into account costs and probabilities (such as [5,32,48]) could be proved polynomial. For any other heuristic which handles a complete model of Decision Tables, counterexamples may be found where the tree generated has an exponential number of leaves.

The problem seems then hopeless for "large" size tables. We must give up Decision Trees for converting Decision Tables, and move back to other less interesting techniques like the rule mask approach [2,7,16].

Fortunately, in most of the practical applications considered until now, Decision Tables remained fairly small. In programming applications for example, a 10 binary conditions, 40 actions table is considered an exceptionally large one in [29]; and [13] analyzing the use of Decision Tables compilers find out that among a large number of tables (118), a vast majority have less than 10 conditions. In pattern recognition applications [1,9], the maximum number of conditions considered is also in the range of 10 to 20, whereas the number of actions has a limit between 50 to 100.

Thus, we may still afford to use for the optimization of Decision Tables algorithms exponentially upper bounded in the worst case, particularly if their average behavior is polynomial. We present in the remainder of this section such a procedure. It is based on a Branch and Bound method proposed initially by [39,40] for expanded limited-entries decision tables, and generalized later. We first introduce the general Branch and Bound algorithm of [18], then the search representation proposed by [24], and finally the approximation scheme is defined and some of its interesting features and properties are presented. The section ends with some results from computer runs on randomly generated data, results which give hints of the average behavior of this approximation scheme.

4.1 An Exact Algorithm for the DTC Problem

The following algorithm is a Branch and Bound procedure, based on a tree search technique and on an estimate function Φ of the goodness of a partial solution.

Let Λ be a set of partial decision trees, Φ a function mapping Λ into R^+, and $\lambda_v(\tau)$ the set of all the partial trees which expand τ from its open branch v ($\lambda_v(\tau)$ contains as many trees as don't-care conditions in the event associated to v).

Algorithm A2

Input: a decision table T

1. Start: with the empty partial tree $\Lambda \leftarrow \{\tau \leftarrow \tau_0\}$;
2. Do While τ is a partial tree;

2.1 Take an open branch v in τ;

2.2 If the subtable v contains only one action a_k Then Do;

2.2.1 Expand τ by completing the open branch v with a leaf containing a_k

2.2.2 End;

2.3 Else Do;

2.3.1 $\Lambda \leftarrow (\Lambda - \tau) \cup \lambda_v(\tau)$;

2.3.2 $\tau \leftarrow$ the partial tree with the maximum number of nodes among the set $\{\tau \in \Lambda \mid \Phi(\tau) \text{ is minimal}\}$

2.3.3 End;

2.4 End;

3 Output: a decision tree $t \leftarrow \tau$ translating T.

As a general property of the Branch and Bound procedure [19-21], the output of algorithm A2 is an optimal tree if and only if the estimate Φ is a non-decreasing consistent lower bound. In other words Φ must verify:

(i) for any partial tree τ' expanding τ: $\Phi(\tau) \leq \Phi(\tau')$;

(ii) for any complete tree: $\Phi(t) = \Psi(t)$.

Condition (i) implies that $\Phi(\tau)$ is a lower bound of the cost $\Psi(t)$ of all the trees that can be generated by completing τ.

A consistent lower bound for Decision Trees is based on the probability q_j to reach an action in a decision table T without evaluating the condition x_j. Such a probability q_j is computed by adding the probabilities of all the events of T for which x_j is either an explicit don't-care condition or a consensus variable relative to some other dependency events or decision events mapped to the same action.

For example in the table of figure 1 we have:

$$
\begin{aligned}
q_2 &= \Pr[(0,\phi,0,0)] + \Pr[(1,1,0,0) \vee (1,0,0,0)] \\
&\quad + \Pr[(1,1,1,0) \vee (1,0,1,0)] \\
&= .2 + 1/2 \times .1 + 1/2 \times .1 = .3
\end{aligned}
$$

Those are the only events which can have a don't-care on x_2.

The conditional probability $q_{j/v}$ to reach an action without evaluating x_j, given the value of the known conditions at the open branch v, can be computed similarly.

For example, if the event corresponding to v is $(1,1,\phi,\phi)$ then:

$$
\begin{aligned}
q_{4/v} &= \Pr[(1,1,0,\ )] + \Pr[(1,1,1,0) \vee (1,1,1,1)] \\
&= .1 + 1/2 \times .1 = .15
\end{aligned}
$$

Those are the only events having $x_1 = x_2 = 0$ and a don'tcare on x_4.

The computation of such $q_{j/v}$ is considerably simplified by defining from a decision table T a <u>consensus array</u> which contains for each condition x_j all the events which may have x_j as a don't-care condition.

For the table of Figure 1, the consensus array is given in figure 6:

	$x_1 = \phi$						$x_2 = \phi$			$x_3 = \phi$						$x_4 = \phi$					
	.1	.15	.1	.3	.1	.1	.05	.05	.2	.2	.1	.05	.15	.2	.15	.1	.15	.05	.1	.25	.15
x_1	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	1	1	0	0	1	1	0	0	1	1	0	1	0	0	1
x_2	1	1	1	0	0	0	ϕ	ϕ	ϕ	1	1	1	0	0	0	1	1	1	0	0	0
x_3	1	0	1	0	0	1	0	1	0	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	0	1	1	1	0	0
x_4	0	1	1	1	0	ϕ	0	0	0	1	0	1	0	1	1	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
	a_1			a_2			a_1		a_2	a_1			a_2			a_1			a_2		

Figure 6

From this array we have directly q_j by summing over the event corresponding to x_j. Thus:

$$q_1 = .85 \ ; \quad q_2 = .3 \ ; \quad q_3 = .85 \ ; \quad q_4 = .8$$

For a branch v, $q_{j/v}$ is given by summing, with an appropriate weight, the event corresponding to x_j and overlapping with v. If v corresponds to the event $(\phi,0,\phi,1)$:

$$q_{1/v} = .3 + 1/2 \times .1 = .35 \ ; \quad q_{3/v} = .2 + .15$$

The following definition of estimate Φ has been proved [8] to lead to a consistent lower bound:

(i) for the empty partial tree τ_0: $\Phi(\tau_0) = \sum_{j=1}^{N} \rho_j (1-q_j)$

(ii) for a partial tree τ' expanding a tree τ by adding to the open branch v a node labelled x_j: $\Phi(\tau') = \Phi(\tau) + \rho_j \times q_{j/v}$

4.2 The And/Or Search Graph

We remark that the same event and subtable can be associated to many open branches of different partial trees. For example the starred (*) branches of the 2 trees of figure 7 correspond both to the event $(1,1,\phi,\phi)$ and then have the same subtable.

```
< x2 >→→→→< x1 >→→→→*        < x1 >→→→→< x2 >→→→→*
   ↓          ↓                 ↓          ↓
   ↓          ↓                 ↓          ↓
```

Figure 7

This is due to the fact that the order in which the conditions are crossed in an open branch does not appear in the corresponding event. From this important remark, [24] designed a very nice representation which reduces the search tree of the previous algorithm in the form of an <u>And/Or graph</u>.(1)

The vertices of this graph are the 3^N composite events hierarchized into (N+1) levels. For $0 \leq k \leq N$, level k contains all the $\binom{N}{k} \times 2^k$ events which test k conditions and have (N-k) don't-cares. Each vertex-

(1) To avoid confusion, in the following edges and vertices refer to the search graph, and nodes and branches to decision trees.

event at level k corresponds to k! open branches of different partial trees. Any of these branches can be expanded in (N-k) different ways by testing one of the (N-k) don't-care conditions. Thus (N-k) Or-edges, called <u>connectors</u>, are issued from a vertex at level k. Each connector corresponds to a don't-care condition, and is divided into 2 And-edges, one for each value of this binary condition.

For example figure 8 shows the edges issued from vertices (ϕ,ϕ,ϕ,ϕ), $(1,\phi,\phi,\phi)$ and $(\phi,0,\phi,\phi)$ of an And/Or graph on 4 conditions.

By similarity with the definitions of section 2, in an And/Or graph corresponding to a decision table T, a vertex whose event overlaps with only one action in T is said to be a <u>terminal vertex</u>. If we start at level zero of the graph, choose recursively at each vertex one connector, follow its 2 And-edges, and stop only at terminal vertices, we will generate a decision tree translating T.

Let us now define the cost structure of our search graph. A connector issued from a vertex v and corresponding to a condition x_j costs $\rho_j \times q_{j/v}$. It is easy to see that the sum of this costs over all the connectors defining a partial tree τ is: $\Phi(\tau) - \Phi(\tau_0)$.

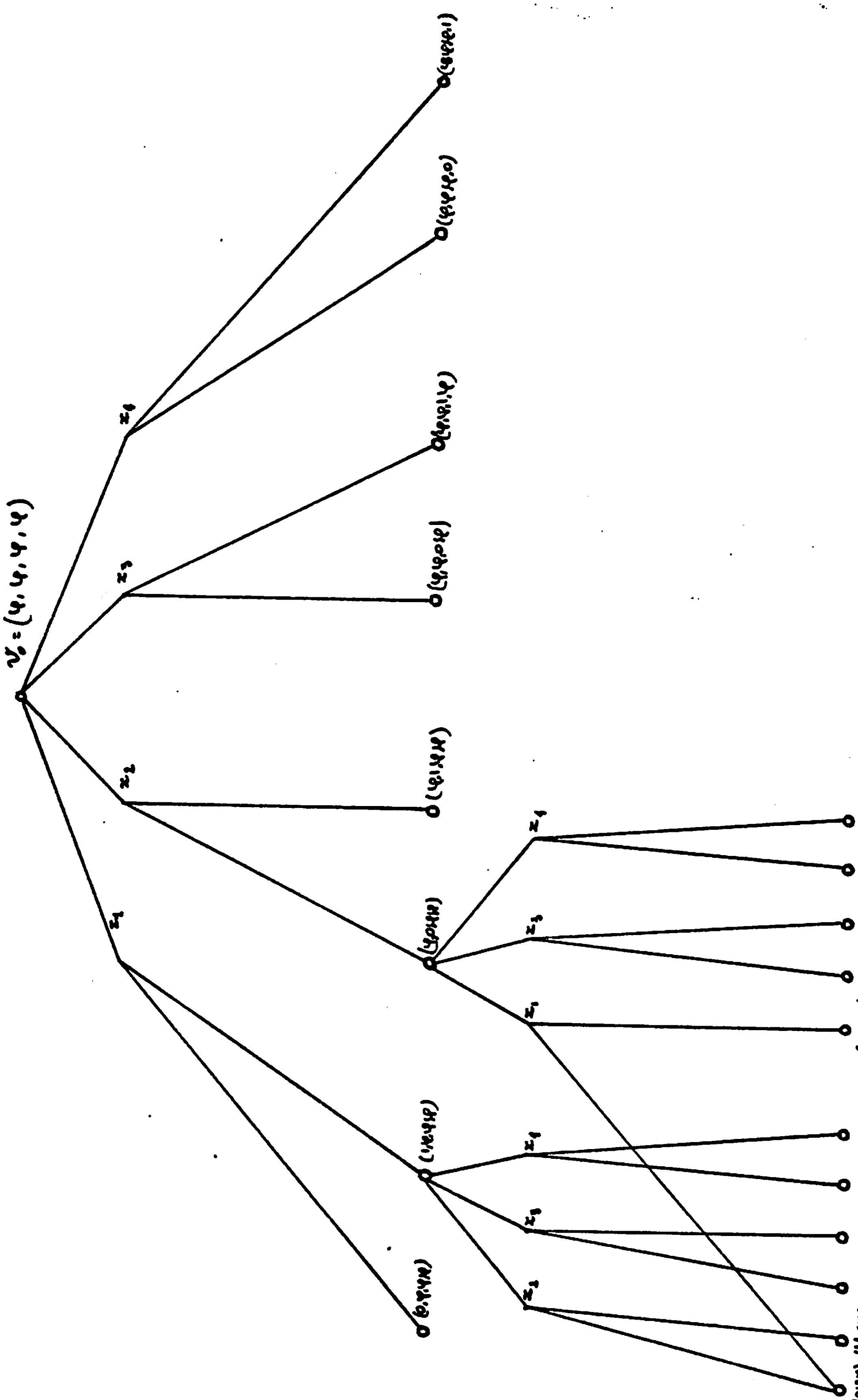


Figure 8

The cost of a vertex v is defined recursively by:

(i) if v is a terminal vertex, cost(v) = 0;

(ii) otherwise, $\text{cost}(v) = \min[\rho_j \times q_{j/v} + \text{cost}(v_{j_1}) + \text{cost}(v_{j_2})]$ over all the connectors issued from v; v_{j_1} and v_{j_2} being the 2 vertices successor of v along the connector labelled x_j.

The connector corresponding to the minimum of this expression is said to be the minimal connector issued from v (one is arbitrarily chosen if many lead to the same minimal cost).

Since Φ is a consistent lower bound, by tracing down from vertex v_0 the minimal connectors (i.e. starting at v_0 and following recursively the minimal connector at each vertex), we will define an optimal tree t* translating a table. Furthermore, the mean decision cost of t* is given by the cost of vertex $v_0 = (\phi,\phi,\ldots,\phi)$: $\Psi(t^*) = \Phi(\tau_0) + \text{cost}(v_0)$.

The following algorithm, due to [24], generates an And/Or search graph corresponding to a decision table T, and thus converts T into an optimal tree. The graph is built progressively. At some step, the <u>expanded vertices</u> are those who already have <u>all</u> their successors. Non-expanded vertices have temporarily a null cost.

<u>Algorithm A3</u>

<u>Input</u>: a decision table T

1. <u>Start</u>: P ← {v_0, the vertex at level 0 of the graph};

2. <u>Do While</u> P contains at least a non-terminal vertex;

2.1 Take a non-terminal vertex v in P;

2.2 Expand v by generating all its non previously existing successors, and assigning to these new vertices a null cost;

2.3 $A \leftarrow \{v\}$;

2.4 Do Until A is empty;

2.4.1 Take in A a vertex v' at the maximum level of the graph and remove v' from A;

2.4.2 Compute the new cost of v' and define its new minimal connector;

2.4.3 If the cost of v has been changed, Then add to A all the predecessors of v';

End;

2.5 $P \leftarrow$ all the non-expanded vertices obtained by tracing down from vertex v_0 the minimal connectors;

2.6 End;

3 $t \leftarrow$ tree obtained by tracing down from v_0 the minimal connectors;

Output: t an optimal tree translating T.

For a proof of this algorithm we may either use the general results of Branch and Bound procedures, or take the Dynamic Programming approach for additive And/Or graph suggested in [24]. In this last reference, it is also shown that algorithm A3 needs in the worst case $0(5^N)$ steps.

4.3 An Approximation Scheme

Any Branch and Bound algorithm can be generalized to an approximation scheme [20,38]. This is indeed a very important feature: a guaranteed approximation is always better than a heuristic one. Furthermore, since no polynomial time heuristic algorithm is known for our problem, an approximation scheme may be, for reasonable values of ε, as fast as a good heuristic.

We develop hereafter a generalization of algorithm A3 to an approximation scheme.

In algorithm A3, each iteration updates the cost of vertices of the graph, changes the minimal connectors, and restarts at vertex v_0. This is in fact a backtracking in the search. For the approximation scheme, the idea is to keep on expanding the graph along the same set of initially minimal connectors, delaying the backtracking as long as the actual cost with these connectors does not depart "too much" from the last lower bound available.

More formally, at some point of the development of the graph, for the vertex at level 0 we define:

$$c = \text{cost}(v_0) = \min_{j=1 \text{ to } N} \{\rho_j \times q_j + \text{cost}(v_{j_1}) + \text{cost}(v_{j_2})\};$$

c' = the next minimal value of the above expression, i.e.

$\min_{j \neq j_0} \{\rho_j \times q_j + \text{cost}(v_{j_1}) + \text{cost}(v_{j_2})\}$, where j_0 corresponds to the minimal connector of v_0.

$$\Delta = \varepsilon(\Phi(\tau_0) + c') + c' - c \text{ for some } \varepsilon \geq 0.$$

The approximation scheme proceeds as follows:

<u>Approximation Scheme AS</u>

<u>Input</u>: a decision table T and a parameter $\varepsilon \geq 0$

1. <u>Start</u>: Expand vertex v_0 by generating all its successors and assigning to them a null cost, compute c, c', $\Delta \leftarrow \varepsilon[\Phi(\tau_0) + c'] + c' - c$, and $\delta \leftarrow \Delta/[\Phi(\tau_0) + c']$, define $P \leftarrow$ {non-terminal successors of v_0 along its minimal connector};

2. <u>Do Until</u> P is empty;

2.1 Take and remove from P a vertex v;

2.2 Expand v by generating its non previously existing successors, and assign to these new vertices a null cost;

2.3 Compute the new cost and define the minimal connector of v;

2.4 Decrease Δ by: $\Delta \leftarrow \Delta - [\text{new cost}(v) - \text{old cost}(v)]$;

2.5 If $\Delta \geq 0$ Then Do;

2.5.1 $P \leftarrow P \cup \{$non-terminal successors of v along its minimal connector$\}$;

2.5.2 $\delta \leftarrow \min\{\delta, \Delta/[\Phi(\tau_0) + c']\}$;

2.5.3 End;

2.6 Else Do;

2.6.1 $A \leftarrow \{$the vertices ancestor of $v\}$;

2.6.2 Do Until A is empty;

2.6.2.1 Take and remove from A a maximum level vertex v';

2.6.2.2 Compute the new cost and define the new minimal connector of v';

2.6.2.3 If the cost of v' changed, Then $A \leftarrow A \cup \{$the ancestors of $v'\}$;

2.6.2.4 End;

2.6.3 $P \leftarrow \{$the non-expanded and non-terminal vertices of the graph obtained by tracing down from v_0 the new minimal connectors$\}$

2.6.4 Recompute c, c' and $\Delta \leftarrow \varepsilon[\Phi(\tau_0) + c'] + c' - c$;

2.6.5 End

2.7 End

3 Generate a tree t by tracing down from v_0 the last minimal connectors;

4 Define $\varepsilon' \leftarrow \max\{0, \varepsilon - \delta\}$;

Output: ε', and the ε'-optimal tree t translating T.

Proof: We will first show that t is an ε-optimal tree. Let t* be an optimal tree translating T, t the output of algorithm AS with input T and ε, and let c, c' and Δ be the last values of these variables in the algorithm.

At any iteration of the algorithm, the search graph is equivalent to a set of partial trees. By updating the cost of all the vertices and their minimal connectors, and tracing down from v_0 the minimal connectors, we will define the partial tree τ^* with minimal estimate. Thus:

$$\Psi(t^*) \geq \Phi(\tau^*) = \mathrm{cost}(v_0) + \Phi(\tau_0) \ .$$

When the algorithm ends, t does not correspond necessarily to the updated minimal connectors of the graph, since vertices have had new costs without updating; and the cost(v_0) is no longer c, but: $\mathrm{cost}(v_0) \geq c'$, and thus $\Psi(t^*) \geq c' + \Phi(\tau_0)$. Since the algorithm updates all the costs of the graph unless $\Delta \geq 0$, we have:

$$\Psi(t) = \Phi(\tau_0) + c + [\varepsilon(\Phi(\tau_0) + c') + c' - c] - \Delta \leq (1+\varepsilon)(\Phi(\tau_0) + c') \ .$$

It follows that:

$$[\Psi(t)-\Psi(t^*)]/\Psi(t^*) \leq \varepsilon \ .$$

Since t is ε-optimal for input ε, if we show that the algorithm produces exactly the same run (i.e. develops the same vertices in the same order) with input ε' as with input ε, our proof will be completed.

Let Δ correspond to the run with input ε, and Δ' to the run with ε'. The two runs start identically until step 2.4 where the sign of Δ and Δ' are checked. Since

$$\varepsilon' \leq \varepsilon \ \Rightarrow \ \Delta' \leq \Delta \ ;$$

furthermore

$$\varepsilon' \geq \varepsilon-\delta \ \Rightarrow \ \Delta' \geq \Delta - \delta(\Phi(\tau_0) + c') \geq \Delta - (\Phi(\tau_0) + c')\min\{\frac{\Delta}{\Phi(\tau_0)+c'}\}$$

Consequently Δ and Δ' have the same sign, and the two runs proceed identically: t is an ε'-optimal tree. □

We remark that:

(i) for $\varepsilon = 0$, t is an optimal tree and AS is then an exact optimization algorithm;

(ii) for $\varepsilon \geq \varepsilon_0 = [\sum_{j=1}^{N} \rho_j - \Phi(\tau_0)]/\Phi(\tau_0)$, AS keeps expanding along the same set of connectors and never updates the graph. This procedure with no backtracing corresponds in fact to the heuristic proposed in [24], but has a decisive advantage over the heuristic: it gives a guarantee ε' of the goodness of the generated tree. As our running experiments suggest, this guarantee has a large practical interest.

Let us now illustrate the algorithm AS by developing a running example on the table of figure 1 with $\varepsilon = .25$. (Refer to figure 6 also for the computation of q_j.) We first compute $\Phi(\tau_0)$:

$$\Phi(\tau_0) = \sum_{j=1}^{4} \rho_j(1-q_j) = 10 \times (1-.85) + 50 \times (1-.3) + 30 \times (1-.85) + 20 \times (1-.8) = 45.$$

The vertex v_0 is expanded, the cost of its connectors are computed by $\rho_j \times q_j$, its successors have a null cost, thus:

$v_0 = (\phi,\phi,\phi,\phi)$; $\mathrm{cost}(v_0) = \min(8.5,15,25.5,16) = 8.5$; x_1 is minimal connector; $P \leftarrow \{(0,\phi,\phi,\phi),(1,\phi,\phi,\phi)\}$ and

$\Delta \leftarrow .25(45+15)+6.5 = 21.5$

The next node to be expanded will be v_1:

$v_1 = (0,\phi,\phi,\phi)$; $\mathrm{cost}(v_1) = \min(10,16.5,10) = 10$; x_2 is minimal connector; $P \leftarrow \{(1,\phi,\phi,\phi),(0,1,\phi,\phi)\}$ since $(0,0,\phi,\phi)$ is terminal; $\Delta \leftarrow 22.5-10 = 11.5$.

$v_2 = (1,\phi,\phi,\phi)$; $\mathrm{cost}(v_2) = \min(5,9,6) = 5$; x_2 is minimal connector, no vertex is added in P since $(1,0,\phi,\phi)$ and $(1,1,\phi,\phi)$ are both terminal; $\Delta \leftarrow 6.5$.

$v_3 = (0,1,\phi,\phi)$; $\text{cost}(v_3) = \min(6,3) = 3$; x_4 is minimal connector; $P \leftarrow \{(0,1,\phi,0)\}$ since $(0,1,\phi, 1)$ is terminal, $\Delta = 3.5$.

$v_4 = (0,1,\phi,0)$; $\text{cost}(v_4) = \min(0) = 0$, x_3 is minimal connector, $\Delta \leftarrow 3.5$.

P is empty. The algorithm ends with the following tree (figure 9) and ε':

$$\varepsilon' = \varepsilon - \min\{\Delta/[\Phi(\tau_0) + c']\} = .25 - 3.5/60 = .192$$

```
‹ x1 ›→→→→‹ x2 ›→→→→□: A1
  ↓         ↓
  ↓         ↓
  ↓         □: A2
  ↓
  ↓
‹ x2 ›→→→→‹ x4 ›→→→→□: A1
  ↓         ↓
  ↓         ↓
  ↓       ‹ x3 ›→→→→□: A1
  ↓         ↓
  ↓         ↓
  ↓         □: A2
  ↓
  ↓
  □: A2
```

Figure 9

The cost of this tree t_1 is:

$$\Psi(t_1) = \Phi(\tau_0) + c + [\varepsilon[\Phi(\tau_0) + c'] + c' - c] - \Delta = 45 + 8.5 + 22.5 - 4.5$$
$$= 71.5$$

This can be verified directly on the tree by:

$$\Psi(t) = 60 \times .35 + 110 \times .1 + 110 \times .05 + 80 \times .2 + 60 \times .15 + 60 \times .15$$
$$= 71.5$$

If we try to converse the table of figure 1 with $\varepsilon = .17$, the initial value of Δ would be 16.7, and after expanding vertex $v_3 = (0,1,\phi,\phi)$ we will have $\Delta = -1.3$. Algorithm AS will enter the backtracking loop (step 2.6) and proceed as follows:

$A \leftarrow \{v_1\}$

Updating of v_1: $cost(v_1) = min(10+3,16.5,10) = 10$;

x_4 is minimal connector

$A \leftarrow \{v_0\}$

Updating of v_0: $cost(v_0) = min(8.5+10+5,15,25.5,16)$;

x_2 is minimal connector

A is now empty: $P \leftarrow \{(\phi,1,\phi,\phi)\}$ since $(\phi,0,\phi,\phi)$ is terminal; and

$\Delta \leftarrow .17 \times (45+16) + 16 - 15 = 11.37$.

This ends the updating loop. The algorithm progresses by expanding the next node in P:

$v_5 = (\phi,1,\phi,\ )$; $cost(v_5) = min(3.5+3,10.5,6) = 6$; x_4 is minimal connector; and $P \leftarrow \{(\phi,1,\phi,0)\}$ since $(\phi,1,\phi,1)$ is terminal; $\Delta \leftarrow 11.37 - 6 = 5.37$.

$v_6 = (\phi,1,\phi,0)$; $cost(v_6) = min(1,3) = 1$; x_1 is minimal connector; $P \leftarrow \{(0,1,\phi,0)\}$ because $(1,1,\phi,0)$ is terminal; $\Delta \leftarrow 4.37$.

$v_7 = (0,1,\phi,0)$; $cost(v_7) = min(0) = 0$; x_3 the only connector is minimal; both $(0,1,0,0)$ and $(0,1,1,0)$ are terminal; thus P is empty and AS outputs the tree t_2 of figure 10 with ε':

$\varepsilon' = .17 - 1.7/60 = .1416$.

The cost of this tree is:

$$\Psi(t_2) = 45 + 15 + 11.37 - 4.37 = 67$$

```
< X2 >→→→→< X4 >→→→→□: A1
  ↓          ↓
  ↓          ↓
  ↓        < X1 >→→→→□: A1
  ↓          ↓
  ↓          ↓
  ↓        < X3 >→→→→□: A1
  ↓          ↓
  ↓          ↓
  ↓          □: A2
  ↓
  ↓
  □: A2
```

Figure 10

An optimal tree translating this table is given in figure 2.

In the approximation scheme AS, as well as in algorithms A1, A2 and A3, the step 2.1 is concerned with the choice of the next vertex to expand. There are two possible alternatives:

(i) take the lower level vertex in P,

(ii) take the higher level vertex.

In order to develop a smaller search graph and to save backtrackings, it is necessary to discriminate between partial trees as early as possible. Since generally the lower is the level of a vertex v, the higher are the values $q_{j/v}$, and then the higher is the cost(v), the first alternative is more efficient.

A third alternative would be to compute the cost of all the vertices in P and to expand the most costly one. This is not very interesting because computing the cost of a vertex is almost as time consuming as expanding it. Furthermore, this alternative introduces a non systematic way of developing the search and complicate the implementation of the algorithm.

Aimed at improving AS we designed two particular strategies. The first strategy develops completely the search graph until level K (practically K = 1 or 2) by expanding all the $\sum_{i=0}^{K} \binom{N}{i} \times 2^i$ first vertices, and then proceeding as in AS. This strategy saves a large number of backtrackings, and although one running experiment suggests that the saving is not compensated by the initial development of the graph, it is worthwhile to be considered in case of very "flat" cost and probability distributions.

The second strategy is concerned with shortening the backtracking loop: instead of updating the entire graph until vertex v_0, we may increase Δ appropriately after each updating, and stop the backtracking as soon as Δ becomes again positive. In this strategy the algorithm continues to develop the same partial tree changing only some of its lower nodes. The length of each backtrack is restrained, but the number of backtracks is augmented. Besides that, in this strategy we need to keep track of the updated and non-updated vertices, which considerably complicates the implementation.

4.4 Some Properties of Algorithm AS

The main advantage of Decision Trees over other techniques of conversion of Decision Tables is that in a tree only the most efficient conditions are evaluated. The tree solves automatically the problem of selection of a subset of "good" conditions, a problem which arises when the table contains many more conditions than necessary.

A condition x_j is redundant in a table T, if by removing x_j from T we have no loss of information. In other words, x_j is redundant in T if we remove from T x_j, the j^{th} component of all the events, and all the resulting inconsistent events (mapped to two different actions), and T is still a complete table.

Consider the example of figure 11. The table T', obtained by removing x_3 from T is still complete: x_3 is redundant. But the trees t_1 and t_2, both translating T are such that: $\Psi(t_2) < \Psi(t_1)$, although t_2 uses the redundant condition x_3. It follows that we may miss the optimal tree by removing the redundant condition before converting a table.

T	.1	.2	.3	.4					
x_1	1	1	0	0	1	1	0	0	10
x_2	1	0	0	1	1	0	0	1	10
x_3	0	0	0	0	1	1	1	0	10
	a_1	a_2	a_3	a_4	b				

T'	.1	.2	.3	.4	
x_1	1	1	0	0	10
x_2	0	1	0	1	10
	a_1	a_2	a_3	a_4	

t_1 :

```
< X1 >→→→→< X2 >→→→→□: A1
  ↓           ↓
  ↓           ↓
  ↓           □: A2
  ↓
  ↓
< X2 >→→→→□: A4
  ↓
  ↓
  □: A3
```

t_2

```
< X3 >→→→→□: A4
  ↓
  ↓
< X1 >→→→→< X2 >→→→→□: A1
  ↓           ↓
  ↓           ↓
  ↓           □: A2
  ↓
  ↓
  □: A3
```

Figure 11

Let us define a <u>fully redundant</u> condition x_j as one which can be removed from a table T without loss, neither of information nor of optimality, for any given cost and probability distributions in T.

If v_j and v_j' are the two subtables corresponding to the two events where $x_i = \phi$ for $i \neq j$; and $x_j = 0$ and 1 respectively, we have the following result:

Lemma: x_j is a fully redundant condition in T if and only if one of the following occurs:

(i) v_j and v_j' are two identical subtables;

(ii) one of the two subtables v_j or v_j' has no decision event (only dependencies).

Proof (Direct part): It is obvious that any x_j verifying (i) or (ii) is redundant. Let us assume that an optimal tree t* translating T tests x_j. If x_j verifies (ii) then from the node of t* labelled by x_j only one branch is issued since the other branch leads to an empty set of decision events. By removing the node x_j from t* we will still reach the same set of leaves but have a lower cost tree: t* is not optimal. If x_j verifies (i), the two subtables corresponding to the two branches of node x_j are identical independently of where node x_j appears in t*. The two subtrees issued from node x_j have the same set of leaves (they may have distinct costs). By removing from t* node x_j and linking its ancestor to its subtree of minimal cost, we still reach the same set of leaves but have a lower cost tree. Thus again t* is not optimal.

(Converse part): Let x_j be completely redundant, and t a tree having x_j as its root. For any cost and probability distributions, there exists a tree t* which does not test x_j and is better than t.

Since $\Psi(t^*) \leq \Psi(t)$ regardless of the values of the parameters, $\Psi(t)$ must contain at least all the (formal) terms of $\Psi(t^*)$. This implies that t* is a subtree of t. Any node in t which does not appear in t* can be removed with no loss of information. But a node cannot be removed unless it has only one branch, or its two subtrees have the same set of leaves and thus can shrink together. Root x_j of t does not appear in t*, and since it can be removed either (i) or (ii) is verified. □

From our definition of q_j, it is straightforward that:

(1) x_j is redundant if and only if $q_j = 1$

(2) x_j is fully redundant if and only if one of the following is true:

(i) $q_j = 1$ and no dependency event appears in the consensus array corresponding to x_j

(ii) $q_j = 1$ and either $x_j = 1$ for all the decision events or $x_j = 0$.

It is also easy to see that if two or more conditions are fully redundant, all of them can be removed from the table without loss. This is not true for simple redundancy.

Notice that x_j may not be redundant in T, and be redundant or fully redundant in a subtable of T.

A minor change in the definition of the consensus array enables us to determine easily and to remove the fully redundant conditions of the initial table or of any of its subtables, each time algorithm AS expands a vertex in the graph.

The basic assumption when decision trees are used for converting decision tables is that the N conditions may be tested in any order. This is not always the case: a condition x_j may not be defined unless $x_i = 1$ for example.

Generally we will define a <u>precedence constraint</u> on a condition x_j by a set of events. One of these events must occur in order to be able to test x_j.

Some simple modifications in algorithm AS lead to decision trees which verify the precedence constraints. Each time a vertex v is expanded, only those conditions which include v in their constraint set have a corresponding connector and may appear later in a tree at this branch.

4.5 Some Experimental Results with Algorithm AS

Algorithm AS has been implemented in an APL system running on the DEC-20 of the University of California at Berkeley Computer Center. The three different strategies presented previously were programmed with the lower level vertex alternative for the development of the search graph.

A pseudo-random generator of decision tables has been designed. It takes as input N, the number of conditions, p, the proportion of dependencies in the table, and r, a maximum ratio between parameters. It then:

(i) defines $d = \lfloor N \times 2^{p+1-N} \rfloor$, d = number of events in D;

(ii) generates e uniformly in [N,3N], $e = |E|$; and M uniformly in [2,M];

(iii) generates e+d events by taking randomly an event v in a set S and expanding it for a randomly chosen condition among its don't-cares; S is set initially to the event $v_0 = (\phi,\phi,\ldots,\phi)$;

(iv) distributes randomly the events of S among the M actions and the set D;

(v) generates N cost parameters $(\rho_1,\ldots,\rho_N)$ uniformly distributed in [1,r] and e probabilities uniformly in [1,r] and normalized.

The decision tables thus generated are consistent and complete.

Algorithm AS has been used on random tables with 5 to 10 conditions. After each call of AS, the following data were recorded:

(i) the ratio of the cost of the tree generated to the maximum cost (i.e. to $\sum_{j=1}^{N} \rho_j$);

(ii) ε', the improved upper bound of the relative gap to the optimum;

(iii) the total number of vertices of the search graph (denoted u);

(iv) the number of expanded vertices of the search graph (denoted s);

(v) the number of backtrackings of the algorithm (denoted b);

(vi) the CPU time of the call; and

(vii) the number of APL operators interpreted during the call (denoted y).

The goals of such experiments were:

(1) to estimate the amount of computer resources spent at each call of AS versus only the search characteristics (u, s, and b), independently of the particular implementation of the algorithm and the system used;

(2) to characterize the average complexity of the approximation scheme AS for the DTC problem, relative to the size of the table N, M, e, d and to ε;

(3) to characterize ε', the improved upper bound, and ε_r, the real relative gap to the optimum, versus N, M, e, d and ε; and

(4) to compare the efficiency of the different strategies for expanding and updating the search graph.

Although more than 1500 runs of algorithm AS were recorded, only a subset of these goals is actually reached. This is due partly to the large number of variables which intervene in our problem. But the main reason in fact is related to the tremendous difference which can be found between two randomly generated decision tables of the same size. Because of this important variability, any consistent mean must be averaged over a large number of tries.

Not enough experiments were recorded with different numbers of dependency events for each table size in order to characterize the influence of the parameter d. The following results concern only decision tables with independent conditions, and are derived from a set of 12, 12, 11, 9, 4 and 2 different tables with respectively N = 5, 6, 7, 8, 9 and 10 conditions. For each table 10 experiments with different costs and probabilities were recorded. In each experiment, algorithm AS has been called 5 times, with inputs ε_0 (no backtracking), $\varepsilon_1 = 0.1$, $\varepsilon_2 = 0.05$, $\varepsilon_3 = 0.01$, $\varepsilon_4 = 0$; and 5 trees ranging from the heuristic one to the optimal one were generated. Not all the experiments could lead to 5 runs of AS: in the case where the output ε'_i of input ε_i was such that: $\varepsilon_j \leq \varepsilon'_i < \varepsilon_{j+1}$, the calls from ε_{i+1} to ε_j were skipped.

A total of 1147 runs support the following results. The statistics were done with the IBM package STATPACK.

We first found that the number y of APL operators interpreted during a run and the CPU time of this run are correlated with a coefficient better than 0.99. Since the CPU time depends on the actual load of the machine and varies significantly for two calls on exactly the same data, we chose to record our results versus y only.

We also observed that u, the number of vertices of the graph, and s, the number of expanded vertices, were linearly dependent.

We tried then to characterize y by a polynomial of N, s, and b. A multiregression showed that among second degree polynomials of these 3 variables, the best result was given by

$$y = \alpha + N(s + \beta * b)$$

for which the multicorrelation coefficient was R = 0.98. This formula supports the intuitive belief that independently of the implementation, algorithm AS has a complexity in $O(N \times (s+b))$.

With the top backtracking strategy (where each updating brings one back to vertex v_0), the average complexity of algorithm AS versus N and ε is given in figure 12, whereas figure 13 displays the average values of ε' and ε_R. Because of the relatively small number of experiments for N = 9 and 10, we do not have reliable measures for these points.

$y \times 10^{-3}$	N = 5	N = 6	N = 7	N = 8
ε_0	4	5	7	13
$\varepsilon_1 = 0.1$	7	12	17	52
$\varepsilon_2 = 0.05$	8	13	20	66
$\varepsilon_3 = 0.01$	9	14	24	90
$\varepsilon_4 = 0$	10	17	31	114

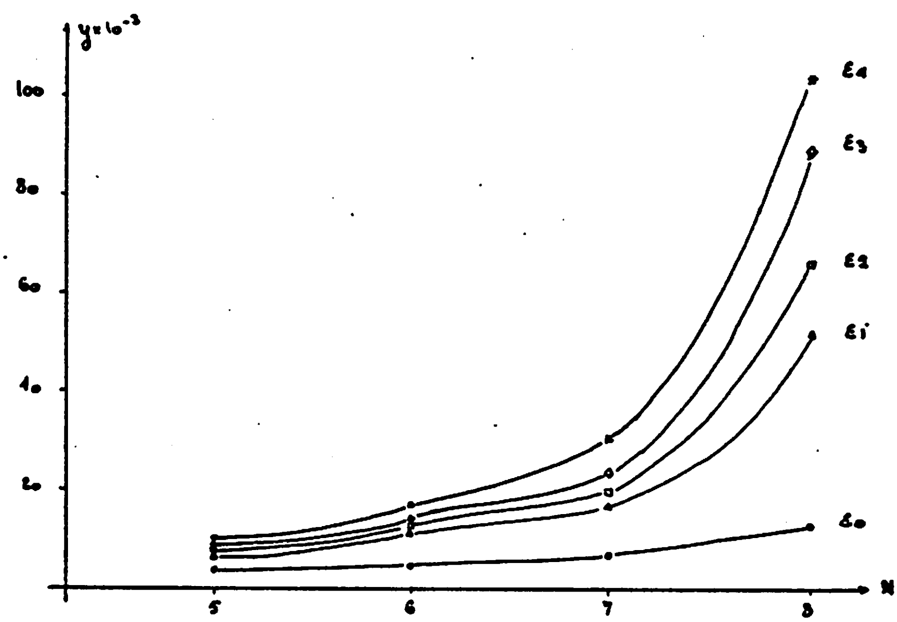


Figure 12

$\varepsilon' \times 100$	N = 5	N = 6	N = 7	N = 8
ε_0	13	19	17	22
$\varepsilon = 0.1$	7.5	8	9	9
$\varepsilon = 0.05$	3	3.7	4.2	4.5

$\varepsilon_R \times 100$	N = 5	N = 6	N = 7	N = 8
ε_0	2	3	4	5
$\varepsilon = 0.1$	0.1	0.2	0.3	1
$\varepsilon = 0.05$	0.05	0.1	0.1	0.2

Figure 13

From figure 12, we can conclude, with the necessary reserves due to the low range of N, that the average behavior of algorithm AS grows from a less than quadratic complexity for $\varepsilon = \varepsilon_0$ (no backtracking), to an almost exponential complexity for $\varepsilon = 0$ (optimal solution). Notice that the gap between $\varepsilon = \varepsilon_0$ and $\varepsilon = 0.1$ is as important as the gap between $\varepsilon = 0.1$ and $\varepsilon = 0$.

From figure 13 we remark that the improvement of ε' over ε decreases when N increases or when ε decreases. The real relative difference ε_R follows a similar behavior, and seems to be at a constant ratio (around 4) to ε'. This ratio is low enough to give to ε' a practical interest: it is as important to know that a heuristic solution ($\varepsilon = \varepsilon_0$) is in the average at 5% of the optimum as to be sure that a particular heuristic tree cannot be worst than 20% optimal, for example.

Our last results concern the comparison of the 3 strategies of AS. Figure 14 shows the average complexity for N = 7 versus ε for strategy S_1 = top backtracking, S_2 = short backtracking, and S_3 = complete expansion of the graph until level 2 and top backtracking later.

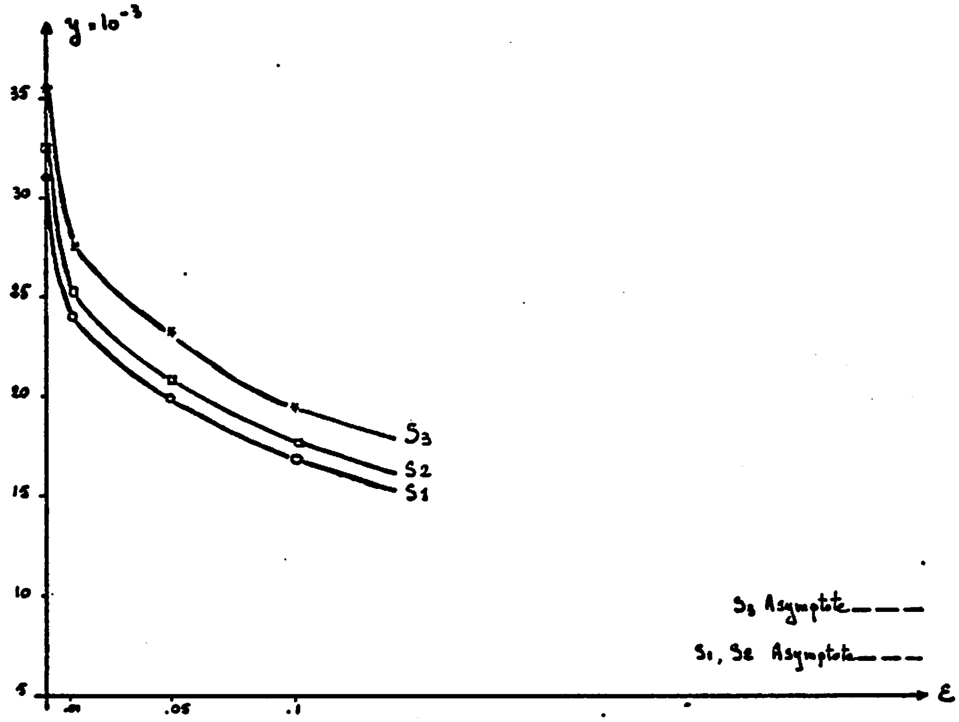


Figure 14

This figure shows clearly the superiority of strategy S_1. For the asymptotic value $\varepsilon = \varepsilon_0$, S_1 and S_2 are equivalent since there is no backtracking.

A side advantage of S_3, which is to give a lower ε', has not been quantified.

5. Conclusion

Compared to Dynamic Programming which needs systematically 3^N steps, and to heuristics which are neither proved polynomial nor guarantee the goodness of their solution, we believe that for converting Decision Tables, the Branch and Bound remains the most interesting method. The And/Or search graph and the ε' feature of algorithm AS improve considerably the advantages of this approach.

If used systematically with ε_0 (no backtracking), the implementation of AS is almost as simple as the implementation of the dynamic programming algorithm, and the computer resources needed are those of a heuristic algorithm.

We conjecture that by assuming some hypothesis only on the cost and probability distributions of a table, it will be possible to prove that AS is in the average polynomial, or polynomial "almost everywhere" following the approach of [15].

In another interesting open problem, each condition has two distinct costs: an evaluation cost and a loading cost. A limited space enables us to put some conditions in the main storage where the loading cost is null. The problem is to find an assignment of the conditions between the main and the secondary storage, and a decision tree converting a table, optimal for the mean decision cost.

We conjecture that if the loading cost of a condition x_j is proportional to the space x_j will occupy if assigned to the main storage, a modified algorithm AS will lead to a solution at a constant bound from the optimum.

Acknowledgments

The author would like particularly to recognize his gratitude to Professor R. M. Karp for the many suggestions, helpful criticisms and the very generous allocation of computer time provided.

To Professor J. P. Jacob, special thanks are expressed for his constant help and guidance during this year.

References

1. Bell, D. A. Decision trees, tables and lattices. In Pattern Recognition, Ideas and Practice. Batchelor, B.G., ed. Plenum Press, New York, 1978, pp. 119-141.

2. Dathe, G. Conversion of decision tables by rule mask method without rule mask. Comm. ACM 15:10 (Oct. 1972).

3. Duncan, G. T. Optimal diagnostic questionnaires. Operations Research 23:1 (Jan. 1975), 22-32.

4. Ganapathy, S. and Rajaraman, V. Information theory applied to the conversion of decision tables to computer programs. Comm. ACM 16:9 (Sept. 1973), 532-534.

5. Garey, M. R. Optimal binary identification procedures. SIAM J. Appl. Math. 23:2 (Sept. 1972), 173-186.

6. Garey, M. R. and Graham, R. L. Performance bounds on the splitting algorithm for binary testing. Acta Informatica 3 (1974), 347-355.

7. Garey, M. R. and Johnson, D. S. Strong NP-completeness results: motivation, examples and implications. J. ACM 25:3 (July 1978), 499-508.

8. Ghallab, M. Consistence, competude et traduction optimale des tables de decision. RAIRO Op. Res. 12:1 (Feb. 1978), 61-84.

9. Ghallab, M. and Giralt, G. A decision method for object identification in robotics. Proc. 17th IEEE C.D.C., San Diego (Jan. 1979).

10. Hughes, M. L., Shank, R. M. and Stern, E. S. Decision Tables. MDI Publications, Wayne, Pennsylvania, 1968.

11. Hyafill, L. and Rivest, R. L. Graph partitioning and constructing optimal decision trees are polynomial complete problems. IRIA Rapport 33, Oct. 1973.

12. Ibramsha, M. and Rajaraman, V. Detection of logical errors in decision table programs. Comm. ACM 21:12 (Dec. 1978), 1016-1025.

13. Jarvis, J. M. An analysis of programming via decision table compilers. SIGPLAN Notices 6:8 (Sept. 1971), 20-32.

14. Karp, R. M. Reducibility among combinatorial problems. In Complexity of Computer Computations. R. E. Miller and J. W. Thatcher, eds. Plenum Press, New York, 1972, pp. 85-104.

15. Karp, R. M. The probabilistic analysis of some combinatorial search algorithms. In Algorithms and Complexity: New Directions and Recent Results. J. Traub, ed. Academic Press, New York, 1976, pp. 1-20.

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

17. King, P. J. H. Ambiguity in limited entry decision tables. Comm. ACM 11:10 (Oct. 1968), 680-684.

18. King, P. J. H. The interpretation of limited entry decision table format and relationships among conditions. Computer J. 12:4 (1969), 320-326.

19. Kohler, W. H. and Steiglitz, K. Characterization and theoretical comparison of branch and bound algorithms for permutation problems. J. ACM 21:1 (Jan. 1974), 140-156.

20. Kohler, W. H. Exact and approximate algorithms for permutation problems. Ph.D. Diss., Princeton University, Princeton, 1972.

21. Lawler, E. L. and Wood, D. E. Branch-and-bound methods: a survey. Op. Res. 14:4 (July 1966), 699-719.

22. Lew, A. Optimal conversion of extended entry decision tables with general cost criteria. Comm. ACM 21:4 (April 1978), 269-279.

23. London, K. Decision Tables: a Practical Approach for Data Processing. Auerbach Publishers, Princeton, New Jersey, 1972.

24. Martelli, A. and Montanari, U. Optimizing decision trees through heuristically guided search. Comm. ACM 21:12 (Dec. 1978), 1025-1039.

25. Maxwell, W. L. The scheduling of single-machine systems: a review. Int. J. Prod. Res. 3:3 (1964), 177-200.

26. McCluskey, G. J. Introduction to the Theory of Switching Circuits. McGraw-Hill Book Co., New York, 1965.

27. Mc Daniel, H. Decision Tables Software. Brandon/Systems Press, Princeton, New Jersey, 1970.

28. Muthukrishnan, C. R. and Rajaraman, V. On the conversion of decision tables to computer programs. Comm. ACM 13:6 (June 1970), 247-251.

29. Myers, H. J. Compiling optimized code from decision tables. IBM J. Res. Develop. 16:5 (Sept. 1972), 489-503.

30. Pitra, C. Theorie des questionnaires. Gauthier-Villars, Paris, 1965.

31. Pitra, C. Graphes et questionnaires. Gauthier-Villars, Paris, 1972.

32. Pollack, S. L. Conversion of limited entry decision tables to computer programs. Comm. ACM 8:11 (Nov. 1965), 677-682.

33. Pollack, S. L., Hicks, H. and Harrison, W. J. Decision Tables: Theory and Practice. Wiley, New York, 1971.

34. Pooch, U. W. Translation of decision tables. Computing Surveys 6:2 (June 1974), 125-151.

35. Press, L. J. Conversion of decision tables to computer programs. Comm. ACM 8:6 (June 1965), 385-390.

36. 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), 339-358.

37. Reinwald, L. T. and Soland, R. M. Conversion of limited entry decision tables to optimal computer programs, II: Minimum storage requirements. J. ACM 14:4 (Oct. 1967), 742-758.

38. Roy, B. Algébre moderne et Theorie des graphes. Dunod, Paris, 1970.

39. Sahni, S. Computationally related problems. SIAM J. Comput. 3:4 (Dec. 1974), 262-279.

40. Sahni, S. and Gonzalez, T. P-complete approximation problems. J. ACM 23:3 (July 1976), 555-565.

41. Schumacher, H. and Sevcik, K. C. The synthetic approach to decision table conversion. Comm. ACM 19:6 (June 1976), 343-351.

42. Schwartz, B. M. LISP 1.5 decision tables implemented for a serial computer and proposed for parallel computers. SIGPLAN Notices 6:8 (Sept. 1971), 93-103.

43. Sethi, I. K. and Chatterjee, B. Efficient decision tree design for discrete variable pattern recognition problems. Pattern Recognition 9 (1977), 197-206.

44. Schwayder, 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.

45. Schwayder, K. Extending the information theory approach to converting limited-entry decision tables to computer programs. Comm. ACM 17:9 (Sept. 1974), 532-537.

46. Veinott, C. G. Programming decision tables in FORTRAN, COBOL, or ALGOL. Comm. ACM 9:1 (Jan. 1966), 31-35.

47. Verhelst, M. The conversion of limited-entry decision tables to optimal and near-optimal flowcharts: two new algorithms. Comm. ACM 15:11 (Nov. 1972), 974-980.

48. Yasui, T. Conversion of decision tables into decision trees. Ph.D. Thesis, Rep. 501, University of Illinois, Feb. 1972.