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
APPRAOXIMATE 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.
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 Tables have been essentially 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: \( \prod \limits_{i \in I} R_i \).

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 adjacent. They differ only by one condition, \( x_4 \), which is their consensus variable. 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 occurrence of any event.

The \( N \) conditions of some given set \( \{x_1, \ldots, x_N\} \) are not necessarily independent. A general formulation of dependency relations 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 occurrence. (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 decision rule is a relation which specifies the action to be taken when some particular event occurs. A decision table 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: \((c_1, \ldots, c_N)\);
- 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:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_1)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(x_2)</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(x_3)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>(x_4)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(a_1)</td>
<td>10.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(a_2)</td>
<td>50.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(D)</td>
<td>30.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(a_3)</td>
<td>20.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 compressed 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 **consistent** 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 referred 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 consistent.

Given a complete and consistent decision table $T = (X, A, D, E, F)$, and a tree $t$ on $X$ and $A$, $t$ translates $T$ 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.

\[
\begin{array}{c}
(2) \rightarrow 1 \\
\downarrow & \rightarrow \\
\downarrow & \rightarrow \\
\downarrow & (4) \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \\
\downarrow & \rightarrow \\
\downarrow & \rightarrow \\
\downarrow & (3) \rightarrow \rightarrow \rightarrow \rightarrow \\
\downarrow & \rightarrow \\
\downarrow & \rightarrow \\
\downarrow & \rightarrow \\
\downarrow & \rightarrow \\
\rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \right

Figure 2

Note: 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: $Z_2 = 50$.
weight: $Pr[(\phi,0,\phi,\phi)] = \frac{1}{2} Pr[(0,\phi,0,0)] + Pr[(0,0,0,1)] + Pr[(0,0,1,\phi)] = .5$

leaf 2: cost: $Z_2 + Z_1 + Z_4 + Z_3 = 110$.
weight: $Pr[(0,1,0,0)] = \frac{1}{2} Pr[(0,\phi,0,0)] = .1$

leaf 3: cost: $Z_2 + Z_1 + Z_4 + Z_3 = 110$.
weight: $Pr[(0,1,1,0)] = \frac{1}{2} Pr[(\phi,1,1,0)] = .05$

leaf 4: cost: $Z_2 + Z_1 + Z_4 = 80$.
weight: $Pr[(0,1,\phi,1)] = .2$

leaf 5: cost: $Z_2 + Z_1 = 60$.
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

$\mathcal{V}(t) = 50 \times .5 + 110 \times .1 + 110 \times .05 + 80 \times .2 + 60 \times .15 = 66.5$

There are $\prod_{k=0}^{N-1} (N-k)^r$ 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 **partial decision tree** $\tau$ is either the empty tree $\tau_0$ 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 **open branch**. 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 \( \tau_0 \) 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 \( v_1 = (\phi, \phi, 0, 0) \)

![Diagram of partial tree](image)

\[
v_1 = (\phi, \phi, 0, 0) \quad v_2 = (\phi, \phi, 1, 0) \quad v_3 = (\phi, 1, \phi, 1)
\]

Figure 3

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

**Algorithm Al.**

**Input:** a decision table \( T \);

1. **Start:** with an empty partial tree \( \tau \subseteq \tau_0 \)
2. **Do While** \( \tau \) is a partial tree;
   1. **Take in** \( \tau \) an open branch \( v \);
   2. **If** subtable corresponding to \( v \) contains only one action \( a_k \) **Then Do**;
      1. **Expand** \( \tau \) by completing the open branch \( v \) with a leaf containing \( a_k \);
      2. **End**;
   3. **Else Do**;
      1. **Choose in** the subtable corresponding to \( v \) a condition \( x_j \);
      2. **Expand** \( \tau \) by appending a node labeled \( x_j \) to the branch \( v \);
      3. **End**;
   4. **End**;
3. **End**;

**Output:** a decision tree \( \tau \subseteq \tau \) translating \( T \).
We remark that:

i) 2.3.1 is a nondeterministic step

ii) if \( \lambda \) is the total number of leaves of the tree generated, algorithm Al has a computing time in \( O(\lambda) \).

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 \( \varepsilon \)-optimal tree translating a table \( T \) if:
  \[
  \frac{\Psi(t) - \Psi(t^*)}{\Psi(t^*)} \leq \varepsilon, \text{ where } t^* \text{ is an optimal tree translating } T.
  \]

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

- \( A(\varepsilon) \) is a polynomial-time approximation scheme 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 fully polynomial-time approximation scheme 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 \( M\leq e \).

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 Node Cover (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].

**Theorem:** The Decision Tables Conversion problem is NP-Hard in the strong sense.

**Proof:** 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 \( T \).

\[
\begin{align*}
\text{G:} & \quad a_5 \\
& \quad x_1 \quad a_1 \quad x_2 \\
& \quad x_4 \quad a_3 \quad x_3 \\
& \quad a_2 \quad a_4
\end{align*}
\]

\[
\begin{align*}
\text{T:} & \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \\
& \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad \phi \quad 1 \quad 1 \quad 1 \quad 1 \\
& \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad \phi \quad 1 \quad 1 \quad 1 \quad 1 \\
& \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad \phi \quad 1 \quad 1 \quad 1 \quad 1
\end{align*}
\]

\[
\begin{align*}
& \quad a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6 \\
& \quad D
\end{align*}
\]

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 $x_1, \ldots, x_k$ be the conditions crossed in the path leading to $a_{M+1}$: The event associated to this leaf $a_{M+1}$ (has $x_1 = x_2 = \cdots = x_k = 0$ and the other conditions are don't-cases) intersects with only action $a_{M+1}$. It follows that the complementary set of events ($x_1 = 1$ or $x_2 = 1$ or $\cdots$ or $x_k = 1$) overlaps with the other actions $a_1, \ldots, a_M$. Thus, in graph G the nodes $x_1, \ldots, x_k$ 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 pseudo-polynomial transformation (T is defined in $O(N^4)$ steps: $N^3$ events each being an N-tuple); and

(ii) Node Cover is not a number problem (the magnitude of the largest number intervening in NC is bounded by N). \qed

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:

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

**Proof:** We use the precedent reduction of the NC problem to the DTC problem. If $t$ is an $\varepsilon$-optimal tree of cost $k$, any other tree $t'$ strictly better than $t$ will cost at most $k-1$. The ratio:
Consider the following table (Figure 5):

<table>
<thead>
<tr>
<th>$T + N_e$</th>
<th>$N_e$</th>
<th>$T - N_e$</th>
<th>$x_e$</th>
<th>$x_e$</th>
<th>$t_e$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>0</td>
<td>$\phi$</td>
<td>$\phi$</td>
<td>$\phi$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>$T$</td>
<td>0</td>
<td>$\phi$</td>
<td>$\phi$</td>
<td>$\phi$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T$</td>
<td>0</td>
<td>$\phi$</td>
<td>$\phi$</td>
<td>$\phi$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>$T$</td>
<td>0</td>
<td>$\phi$</td>
<td>$\phi$</td>
<td>$\phi$</td>
<td>$\phi$</td>
</tr>
</tbody>
</table>

Until now we did not consider a basic question about the complexity which computing time is not only in $O(N/\epsilon^k)$ but also in $O(N/\epsilon)$.

For $\epsilon \leq N/\text{I/e}$, the rest of the argument applies as before.
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 disjoint. 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 $\sum_{1}^{\alpha_1} 2^1$ over all the events of $E \cup D$, $\alpha_1$ being the number of don't-cares of the considered event. The table is complete if $\sum_{1}^{\alpha_1} 2^1 = 2^N$. 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 $\phi$ of the goodness of a partial solution.

Let $\Lambda$ be a set of partial decision trees, $\phi$ a function mapping $\Lambda$ into $\mathbb{R}^+$, and $\lambda_v(\tau)$ the set of all the partial trees which expand $\tau$ 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 $\Lambda \gets \{\tau + \tau_0\}$;
2. Do While $\tau$ is a partial tree;
   2.1 Take an open branch $v$ in $\tau$;
   2.2 If the subtable $v$ contains only one action $a_k$ Then Do;
   2.2.1 Expand $\tau$ by completing the open branch $v$ with a leaf containing $a_k$
   2.2.2 End;
2.3 Else Do;
   2.3.1 $\Lambda \gets (\Lambda - \tau) \cup \lambda_v(\tau)$;
   2.3.2 $\tau \gets$ the partial tree with the maximum number of nodes among the set $\{\tau \in \Lambda \mid \phi(\tau)$ is minimal$\}$
   2.3.3 End;
2.4 End;

3 Output: a decision tree $t \to t$ 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 $\phi$ is a non-decreasing consistent lower bound. In other words $\phi$ must verify:

(i) for any partial tree $t'$ expanding $t$: $\phi(t) \leq \phi(t')$;
(ii) for any complete tree: $\phi(t) = \psi(t)$.

Condition (i) implies that $\phi(t)$ is a lower bound of the cost $\psi(t)$ of all the trees that can be generated by completing $T$.

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:

$$q_2 = \Pr[(0,\phi,0,0)] + \Pr[(1,1,0,0) \lor (1,0,0,0)]$$
$$+ \Pr[(1,1,1,0) \lor (1,0,1,0)]$$
$$= .2 + 1/2 \times .1 + 1/2 \times .1 = .3$$

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:

$$q_{4/v} = \Pr[(1,1,0,\phi)] + \Pr[(1,1,1,0) \lor (1,1,1,1)]$$
$$= .1 + 1/2 \times .1 = .15$$
Those are the only events having $x_1 = x_2 = 0$ and a don't-care on $x_4$.

The computation of such $q_j/v$ is considerably simplified by defining from a decision table $T$ a consensus array 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:

\[
\begin{array}{cccc|cccc|cccc|cccc|cccc}
  x_1 = \phi & x_2 = \phi & x_3 = \phi & x_4 = \phi \\
  \begin{array}{cccc}
  .1 & .15 & .1 & .3 \\
  .1 & .15 & .2 & .15 \\
  .1 & .15 & .15 & .15 \\
  \end{array} &
  \begin{array}{cccc}
  .1 & .05 & .2 & .05 \\
  .2 & .05 & .15 & .2 \\
  .1 & .05 & .15 & .1 \\
  \end{array} &
  \begin{array}{cccc}
  .1 & .05 & .15 & .15 \\
  .1 & .05 & .15 & .15 \\
  .05 & .05 & .2 & .15 \\
  \end{array} &
  \begin{array}{cccc}
  .05 & .05 & .2 & .15 \\
  .1 & .05 & .15 & .2 \\
  .1 & .05 & .15 & .1 \\
  \end{array} &
  \begin{array}{cccc}
  .05 & .05 & .2 & .15 \\
  .1 & .05 & .15 & .1 \\
  ,1 & .05 & .15 & .1 \\
  \end{array} &
  \begin{array}{cccc}
  .05 & .05 & .2 & .15 \\
  .1 & .05 & .15 & .1 \\
  ,1 & .05 & .15 & .1 \\
  \end{array} &
  \begin{array}{cccc}
  .05 & .05 & .2 & .15 \\
  .1 & .05 & .15 & .1 \\
  ,1 & .05 & .15 & .1 \\
  \end{array} &
  \begin{array}{cccc}
  .05 & .05 & .2 & .15 \\
  .1 & .05 & .15 & .1 \\
  ,1 & .05 & .15 & .1 \\
  \end{array} &
\end{array}
\]

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>$a_2$</td>
<td>$a_1$</td>
<td>$a_2$</td>
</tr>
</tbody>
</table>

Figure 6

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

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

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 $(\phi,0,\phi,1)$:

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

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

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

(ii) for a partial tree $\tau'$ expanding a tree $\tau$ by adding to the open branch $v$ a node labelled $x_j$: $\Phi(\tau') = \Phi(\tau) + p_j \times q_j/v$
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,∅,∅) and then have the same subtable.

\[
\begin{array}{c}
\langle x_2 \rangle \longrightarrow \langle x_1 \rangle \longrightarrow * \\
\downarrow \quad \downarrow \\
\downarrow \quad \downarrow \\
\end{array}
\quad \quad \quad
\begin{array}{c}
\langle x_1 \rangle \longrightarrow \langle x_2 \rangle \longrightarrow * \\
\downarrow \quad \downarrow \\
\downarrow \quad \downarrow \\
\end{array}
\]

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 And/Or graph. (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 connectors, 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 $(\phi,\phi,\phi,\phi)$, $(1,\phi,\phi,\phi)$ 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 terminal vertex. 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 $p_j \times q_{j/v}$. It is easy to see that the sum of this costs over all the connectors defining a partial tree $\tau$ is: $\Phi(\tau) - \Phi(\tau_0)$. 
The cost of a vertex $v$ is defined recursively by:

(i) if $v$ is a terminal vertex, $\text{cost}(v) = 0$;

(ii) otherwise, $\text{cost}(v) = \min[p_j \times q_j/v + \text{cost}(v_{j_1}) + \text{cost}(v_{j_2})]$ 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 $\phi$ 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 $v_0 = (\phi, \phi, ..., \phi)$: $\Psi(t^*) = \phi(t_0) + \text{cost}(v_0)$.

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 expanded vertices are those who already have all their successors. Non-expanded vertices have temporarily a null cost.

**Algorithm A3**

**Input:** a decision table $T$

1. **Start:** $P \leftarrow \{v_0\}$, the vertex at level 0 of the graph;

2. **Do While** $P$ 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 + \{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 + all the non-expanded vertices obtained by tracing down from vertex v₀ the minimal connectors;
2.6 End;
3 t + tree obtained by tracing down from v₀ 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 $O(\delta^N)$ 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 $\epsilon$, as fast as a good heuristic.

-25-
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:

\[
c = \text{cost}(v_0) = \min_{j=1}^{N} \{ p_j \times q_j + \text{cost}(v_j_1) + \text{cost}(v_j_2) \};
\]

\[
c' = \text{the next minimal value of the above expression, i.e.}\]

\[
\min_{j \neq j_0} \{ p_j \times q_j + \text{cost}(v_j_1) + \text{cost}(v_j_2) \}, \text{ where } j_0 \text{ corresponds to the minimal connector of } v_0.
\]

\[
\Delta = \epsilon(\Phi(T_0) + c') + c' - c \text{ for some } \epsilon \geq 0.
\]

The approximation scheme proceeds as follows:

**Approximation Scheme AS**

**Input:** a decision table \( T \) and a parameter \( \epsilon \geq 0 \)

1. **Start:** Expand vertex \( v_0 \) by generating all its successors and assigning to them a null cost, compute \( c, c' \),
   \( \Delta = \epsilon(\Phi(T_0) + c') + c' - c \), and \( \delta = \Delta/[\Phi(T_0) + c'] \), define \( P = \{ \text{non-terminal successors of } v_0 \text{ along its minimal connector} \} \);

2. **Do Until** \( P \) 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 \( \Delta \) by: \( \Delta = \Delta - [\text{new cost}(v) - \text{old cost}(v)] \);
2.5 \textbf{If} \( \Delta > 0 \) \textbf{Then Do:}
\hspace{1em} P \leftarrow P \cup \{\text{non-terminal successors of } v \text{ along its minimal 
\hspace{1em} connector}\};
\hspace{1em} \delta \leftarrow \min\{\delta, \Delta/[(T_0) + c']\};
\hspace{1em} \text{End;}
2.6 \textbf{Else Do:}
\hspace{1em} A \leftarrow \{\text{the vertices ancestor of } v\};
\hspace{1em} \text{Do Until } A \text{ is empty;}
\hspace{2em} \text{Take and remove from } A \text{ a maximum level vertex } v';
\hspace{2em} \text{Compute the new cost and define the new minimal 
\hspace{2em} connector of } v';
\hspace{2em} \text{If the cost of } v' \text{ changed, Then } A \leftarrow A \cup \{\text{the 
\hspace{2em} ancestors of } v'\};
\hspace{2em} \text{End;}
\hspace{1em} P \leftarrow \{\text{the non-expanded and non-terminal vertices of the 
\hspace{1em} graph obtained by tracing down from } v_0 \text{ the new 
\hspace{1em} minimal connectors}\}
\hspace{1em} \text{Recompute } c, c' \text{ and } \Delta + \varepsilon(\Phi(T_0) + c') + c' - c;
\hspace{1em} \text{End}
2.7 \text{End}
3 \hspace{1em} \text{Generate a tree } t \text{ by tracing down from } v_0 \text{ the last minimal connectors;}
4 \hspace{1em} \text{Define } \varepsilon' = \max\{0, \varepsilon - \delta\};
\textbf{Output: } \varepsilon', \text{ and the } \varepsilon'\text{-optimal tree } t \text{ translating } T.
Proof: We will first show that \( t \) is an \( \epsilon \)-optimal tree. Let \( t^* \) be an optimal tree translating \( T \), \( t \) the output of algorithm AS with input \( T \) and \( \epsilon \), and let \( c, c' \) and \( \Delta \) 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 \( T^* \) with minimal estimate. Thus:

\[
\Psi(t^*) \geq \Phi(t^*) = \text{cost}(v_0) + \Phi(t^*_0) .
\]

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: \( \text{cost}(v_0) \geq c' \), and thus \( \Psi(t^*) \geq c' + \Phi(t^*_0) \). Since the algorithm updates all the costs of the graph unless \( \Delta \geq 0 \), we have:

\[
\Psi(t) = \Phi(t_0) + c + [\epsilon(\Phi(t_0) + c') + c' - c] - \Delta \leq (1+\epsilon)(\Phi(t_0) + c') .
\]

It follows that:

\[
\frac{[\Psi(t)-\Psi(t^*)]}{\Psi(t^*)} \leq \epsilon .
\]

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

Let \( \Delta \) correspond to the run with input \( \epsilon \), and \( \Delta' \) to the run with \( \epsilon' \). The two runs start identically until step 2.4 where the sign of \( \Delta \) and \( \Delta' \) are checked. Since

\[
\epsilon' \leq \epsilon \Rightarrow \Delta' \leq \Delta ;
\]

furthermore

\[
\epsilon' \geq \epsilon - \delta \Rightarrow \Delta' \geq \Delta - \delta(\Phi(t_0) + c') \geq \Delta - (\Phi(t_0) + c')\min\left\{\frac{\Delta}{\Phi(t_0) + c'}\right\}
\]

Consequently \( \Delta \) and \( \Delta' \) have the same sign, and the two runs proceed identically: \( t \) is an \( \epsilon' \)-optimal tree. \( \square \)
We remark that:

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

(ii) for $\varepsilon \geq \varepsilon_0 = [\sum_{j=1}^{N} \rho_j - \Phi(t_0)]/\Phi(t_0)$, 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 $\varepsilon'$ 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 $\Phi(t_0)$:

\[
\Phi(t_0) = \sum_{j=1}^{4} \rho_j (1-q_j) = 10 \times (1-.85) + 50 \times (1-.3) + 30 \times (1-.85) + 20 \times (1-.8) = 45.
\]

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

\[v_0 = (\phi,\phi,\phi,\phi); \quad \text{cost}(v_0) = \min(8.5,15,25.5,16) = 8.5; \quad x_1 \text{ is minimal connector}; \quad P + \{(0,\phi,\phi,\phi),(1,\phi,\phi,\phi)\} \text{ and } \Delta = .25(45+15) + 6.5 = 21.5\]

The next node to be expanded will be $v_1$:

\[v_1 = (0,\phi,\phi,\phi); \quad \text{cost}(v_1) = \min(10,16.5,10) = 10; \quad x_2 \text{ is minimal connector}; \quad P + \{(1,\phi,\phi,\phi),(0,1,\phi,\phi)\} \text{ since } (0,0,\phi,\phi) \text{ is terminal}; \quad \Delta = 22.5 - 10 = 11.5.\]

\[v_2 = (1,\phi,\phi,\phi); \quad \text{cost}(v_2) = \min(5,9,6) = 5; \quad x_2 \text{ is minimal connector, no vertex is added in } P \text{ since } (1,0,\phi,\phi) \text{ and } (1,1,\phi,\phi) \text{ are both terminal}; \quad \Delta = 6.5.\]
$v_3 = (0, 1, \phi, \phi); \text{cost}(v_3) = \min(6, 3) = 3; \ x_4 \text{ is minimal}

connector; \ P + \{(0, 1, \phi, 0)\} \text{ since } (0, 1, \phi, 1) \text{ is terminal}, \Delta = 3.5.

$v_4 = (0, 1, \phi, 0); \text{cost}(v_4) = \min(0) = 0, \ x_3 \text{ is minimal}

connector, \Delta + 3.5.

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

$$
\varepsilon' = \varepsilon - \min\{\Delta/[(\phi(\tau_0) + c')]\} = 0.25 - 3.5/60 = 0.192
$$

Figure 9

The cost of this tree $\tau_1$ is:

$$
\Psi(\tau_1) = \phi(\tau_0) + c + [\varepsilon[\phi(\tau_0) + c')] + c' - c] - \Delta = 45 + 8.5 + 22.5 - 4.5
\Psi(\tau_1) = 71.5
$$

This can be verified directly on the tree by:

$$
\Psi(\tau) = 60 \times 0.35 + 110 \times 0.1 + 110 \times 0.05 + 80 \times 0.2 + 60 \times 0.15 + 60 \times 0.15
\Psi(\tau) = 71.5
$$

-30-
If we try to converse the table of figure 1 with \( \varepsilon = .17 \), the initial value of \( \Delta \) 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:

\[ A + \{v_1\} \]

Updating of \( v_1 \): cost\((v_1) = \min(10+3,16.5,10) = 10; \]
\( x_4 \) is minimal connector

\[ A + \{v_0\} \]

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: \( P + \{(\phi,1,\phi,\phi)\} \) since \((\phi,0,\phi,\phi)\) is terminal; and
\[ \Delta = .17 \times (45+16) + 16 - 15 = 11.37. \]

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

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

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

\( v_7 = (0,1,\phi,0); \) \( \text{cost}(v_7) = \min(0) = 0; \) \( x_3 \) 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 \( \varepsilon' \):
\[ \varepsilon' = .17 - 1.7/60 = .1416. \]

The cost of this tree is:
\[ \psi(t_2) = 45 + 15 + 11.37 - 4.37 = 67. \]
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 backtracking, 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 backtracks, 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 $\Delta$ appropriately after each updating, and stop the backtracking as soon as $\Delta$ 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.

\[
\begin{array}{cccccc}
T & .1 & .2 & .3 & .4 \\
\hline
x_1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 10 \\
x_2 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 10 \\
x_3 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 10 \\
a_1 & a_2 & a_3 & a_4 & b \\
\hline
\end{array}
\]

\[
\begin{array}{cccccc}
T' & .1 & .2 & .3 & .4 \\
\hline
x_1 & 1 & 1 & 0 & 0 & 10 \\
x_2 & 0 & 1 & 0 & 1 & 10 \\
a_1 & a_2 & a_3 & a_4 & b \\
\hline
\end{array}
\]

Figure 11

Let us define a **fully redundant** 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
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 precedence constraint on a condition $x_j$ by a set of events. One of these events must occur in order to be able to test $x_j$. 

-36-
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 \( d = \left\lfloor N \times 2^{p+1-N} \right\rfloor \), \( d \) = 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 \[ \frac{N}{\sum_{j=1}^{N} \rho_j}; \]

(ii) \( \varepsilon' \), 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 \( \varepsilon \);

3. to characterize \( \varepsilon' \), the improved upper bound, and \( \varepsilon_r \), the real relative gap to the optimum, versus \( N, M, e, d \) and \( \varepsilon \); 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 $e_0$ (no backtracking), $e_1 = 0.1$, $e_2 = 0.05$, $e_3 = 0.01$, $e_4 = 0$; 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 $e'_i$ of input $e_i$ was such that: $e_j < e'_i < e_{j+1}$, the calls from $e_{i+1}$ to $e_j$ 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

$$y = a + N(s + \beta b)$$

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 $\varepsilon$ is given in figure 12, whereas figure 13 displays the average values of $\varepsilon'$ and $\varepsilon_R$. Because of the relatively small number of experiments for $N = 9$ and $10$, we do not have reliable measures for these points.

<table>
<thead>
<tr>
<th>$\gamma \times 10^{-3}$</th>
<th>$N = 5$</th>
<th>$N = 6$</th>
<th>$N = 7$</th>
<th>$N = 8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon_0$</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>13</td>
</tr>
<tr>
<td>$\varepsilon_1 = 0.1$</td>
<td>7</td>
<td>12</td>
<td>17</td>
<td>52</td>
</tr>
<tr>
<td>$\varepsilon_2 = 0.05$</td>
<td>8</td>
<td>13</td>
<td>20</td>
<td>66</td>
</tr>
<tr>
<td>$\varepsilon_3 = 0.01$</td>
<td>9</td>
<td>14</td>
<td>24</td>
<td>90</td>
</tr>
<tr>
<td>$\varepsilon_4 = 0$</td>
<td>10</td>
<td>17</td>
<td>31</td>
<td>114</td>
</tr>
</tbody>
</table>
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 $\epsilon = \epsilon_0$ (no backtracking), to an almost exponential complexity for $\epsilon = 0$ (optimal solution). Notice that the gap between $\epsilon = \epsilon_0$ and $\epsilon = 0.1$ is as important as the gap between $\epsilon = 0.1$ and $\epsilon = 0$.

From figure 13 we remark that the improvement of $\epsilon'$ over $\epsilon$ decreases when $N$ increases or when $\epsilon$ decreases. The real relative difference $\epsilon_R$ follows a similar behavior, and seems to be at a constant ratio (around 4) to $\epsilon'$. This ratio is low enough to give to $\epsilon'$ a practical interest: it is as important to know that a heuristic solution ($\epsilon = \epsilon_0'$) 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 $\epsilon$ for strategy $S_1 = \text{top backtracking}$, $S_2 = \text{short backtracking}$, and $S_3 = \text{complete expansion of the graph until level 2 and top backtracking later}$. 

<table>
<thead>
<tr>
<th>$\epsilon' \times 100$</th>
<th>$N = 5$</th>
<th>$N = 6$</th>
<th>$N = 7$</th>
<th>$N = 8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon_0$</td>
<td>13</td>
<td>19</td>
<td>17</td>
<td>22</td>
</tr>
<tr>
<td>$\epsilon = 0.1$</td>
<td>7.5</td>
<td>8</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>$\epsilon = 0.05$</td>
<td>3</td>
<td>3.7</td>
<td>4.2</td>
<td>4.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\epsilon_R \times 100$</th>
<th>$N = 5$</th>
<th>$N = 6$</th>
<th>$N = 7$</th>
<th>$N = 8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon_0$</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>$\epsilon = 0.1$</td>
<td>0.1</td>
<td>0.2</td>
<td>0.3</td>
<td>1</td>
</tr>
<tr>
<td>$\epsilon = 0.05$</td>
<td>0.05</td>
<td>0.1</td>
<td>0.1</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Figure 13
This figure shows clearly the superiority of strategy $S_1$. For the asymptotic value $\epsilon = \epsilon_0$, $S_1$ and $S_2$ are equivalent since there is no backtracking.

A side advantage of $S_3$, which is to give a lower $\epsilon'$, 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 $\varepsilon_0$ (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


-46-


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.


