DEPARTEMENT TOEGEPASTE ECONOMISCHE WETENSCHAPPEN

ONDERZOEKSRAPPORT NR 9602

A Branch and Bound Algorithm to Optimize the Representation of Tabular Decision Processes

by

Jan VANTHIENEN

Elke DRIES

Katholieke Universiteit Leuven

Naamsestraat 69, B-3000 Leuven

ONDERZOEKSRAPPORT NR 9602

A Branch and Bound Algorithm to Optimize the Representation of Tabular Decision Processes

by

Jan VANTHIENEN
Elke DRIES

D/1996/2376/2

1. Introduction

Decision tables (DTs) were originally used as a technique in computer programming. Due to its representational capabilities, its application area has extended later on (up till now) to several other domains with logical complexity, such as: information systems analysis and design, laws and regulations, structuring of management decisions, medical diagnosis, etc. [12] [13] [15] [18] [25]. The most actual application field of DTs is undoubtedly found in the area of knowledge based and expert systems. Because of their ability to detect incompleteness and inconsistency in an easy way, DTs are applied to verify and validate knowledge based systems and to minimize maintenance anomalies [4] [7] [10]. Also some initial efforts are made to use DTs in the knowledge acquisition phase [7] [8] [14] [19]. Furthermore DTs and decision trees are important structures in inductive machine learning [11] [26].

Results with the PROLOGA system [22], a design tool for computer-supported construction, manipulation, validation and optimization of DTs, show the ability to *acquire* and *verify* knowledge in the form of a hierarchy of DTs. Depending on the application, the hierarchy of validated DTs can be automatically transformed into a desired *target representation*, such as text, optimal rules, tables or trees. This target representation can then serve as a basis to automatically *implement* the decision logic. Different implementation strategies can be chosen, such as program code, optimal test sequences and implemented rules. In this way, the full trajectory of the life cycle of the intelligent system is covered as complete as possible. Moreover, the automatic transition between the different formalisms, makes it possible to change between different views of the knowledge and to combine the advantages of different representation formalisms [18] [19].

DTs occur in different formats [6] [21]. In a *single hit* table, each possible combination of condition states is found in exactly one decision column, while in a *multiple hit* table the columns are not exclusive. Single hit tables occur in expanded or contracted form. In the *expanded* table, all combinations of condition states are explicitly enumerated, while in the *contracted* table adjacent columns or groups of columns that only differ in the state value of one condition and that result in the same action configuration are joined, thus minimizing the number of columns. Only expanded single hit DTs have the full capability to check a given specification for completeness, consistency and correctness. Therefore, the expanded single hit table is a suitable representation mechanism to acquire and verify knowledge. Once the expanded DTs are built and validated, they can be contracted in order to optimize the representation. Two types of contracted DTs are possible, depending on the fact whether the order of the conditions remains *fixed* during the contraction process or whether it is *changed* in order to obtain the simplest contracted DT. The overview and readability of optimal contracted DTs make them very useful as target representation. They can serve as a basis for implementation, document implemented intelligent systems or support manual decision making. In this paper an algorithm is presented to contract expanded DTs in an optimal way. The algorithm is described in the context of its automation in the PROLOGA tool.

2. Concepts and purpose of the algorithm

A DT[1] describes a procedural decision situation, characterized by one or more conditions, whose different combinations of states are uniquely related to a combination of action values.

Each condition C_i (i = 1..cnum) consists of a condition subject CS_i, a condition domain CD_i, i.e. the set of all possible values of condition subject CS_i, and a set of condition states $CT_i = \{S_{ik}\}$, k = 1 .. statnum [i], with each condition state a logic expression concerning the elements of CD_i. The elements of CD_i involved in a condition state S_{ik} determine a subset of CD_i, such that the set of all these subsets constitutes a partition of CD_i.
Each action A_j (j = 1..anum) consists of an action subject AS_j and a set of action values AV_j = {true (x), false (-), nil (.)}.

A DT is a function from $CT_1 \times CT_2 \times ... \times CT_{cnum}$ to $AV_1 \times AV_2 \times ... \times AV_{anum}$ such that each possible combination of condition states is mapped into one (completeness) and only one (exclusivity) action configuration.

If each column of the DT contains only one state for each condition (no contractions or irrelevant conditions), the table is called an *expanded* DT. Figure 1 shows an expanded DT, representing the following decision situation, which is based on examination regulations from practice (see [22] for an overview of the construction of DTs with PROLOGA based on specification rules):

> ***Examination Regulations***
>
> *The Board of Examiners autonomously determines for every student the overall result of the examinations, taking into account the provisions of articles 1 to 4. The Board thus decides whether a student*
> - *has passed;*
> - *has not passed, with the possibility to carry over examination results;*
> - *has not passed.*
>
> ***Article 1***
> *A student who has obtained at least 10 points out of 20 for all courses has passed the examination session.*
>
> ***Article 2***
> *A student who has obtained more than 1 insufficient mark, as well as a student who did not obtain at least half of the points for all of the courses taken as a whole, has not passed the examination session. Students that have not passed, have the possibility to carry over examination results for educational units for which they obtained at least 10 points out of 20, provided that marks have been attributed for all of the courses and provided that they have obtained at least half of the points overall or have passed at least half of the courses.*

[1] In the remainder of the text, the term DT only covers the single hit DT.

Article 3
For a student that has obtained at least half of the points overall and one insufficient mark, a vote is taken in order to determine whether he/she has passed the examination session or whether he/she has not passed with the possibility to carry over examination results.

Article 4
A student is proclaimed as not passing if he/she has registered for an examination session and does not participate in the examinations or terminates participation. A student who did not attend all examinations is not allowed to carry over any examination results.

The following conditions (items about which information is needed in order to make a decision) and relevant condition states can be extracted from the text:

1. number of insufficiencies: 0, 1, more;
2. overall points obtained by the student: <half, >=half;
3. number of courses the student has passed: <half, >=half;
4. participation in all examinations: yes, no.

The following actions (possible results of the decision making process) can be extracted from the text:

1. the student has passed;
2. the student has not passed, but has the possibility to carry over examination results;
3. the student has not passed;
4. a vote has to be taken in order to determine whether the student has passed or has not passed with the possibility to carry over examination results.

These conditions and actions are found in shortened form in the left part of the DT. All possible combinations of condition states are found in the upper right part of the DT. Notice that the explicit enumeration of all possible decision cases guarantees completeness and consistency and supports verification and validation of the decision logic. Impossible combinations of condition states are assigned to a supplementary action 'impossible'.

1. insufficiencies	0								1								more							
2. overall points	<half				>=half				<half				>=half				<half				>=half			
3. courses passed	<half		>=half		<half		>=half		<half		>=half		<half		>=half		<half		>=half		<half		>=half	
4. attended all exams	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N
1. passed	.	.	.	.	.	.	x	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
2. n passed carry over	.	.	.	.	.	.	.	.	.	.	x	.	.	.	.	.	.	.	x	.	x	.	x	.
3. not passed	.	.	.	.	.	.	.	x	.	.	.	x	.	.	.	x	x	x	.	x	.	x	.	x
4. vote	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	.	.	.	.	.	.	.	.	.
5. impossible	x	x	x	x	x	x	.	.	x	x	.	.	x	x	.	.	.	.	.	.	.	.	.	.
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

Figure 1: Expanded DT

Once an expanded DT is validated, it can be simplified. The decision columns of a contracted DT can contain a complete or partial contraction of condition states for one or more conditions. A contracted DT can be constructed from an expanded DT in the following way (see [20] for details on the contraction algorithms in PROLOGA): adjacent columns or groups of columns that only differ in the state value of one condition and that have equal action configurations can be contracted into respectively one combined column or one combined group of columns. If all condition states of the concerning condition can be combined (*complete contraction*) , it becomes irrelevant, which is denoted by means of a don't care entry. It is however also possible that only a limited number of adjacent states can be combined (*partial contraction*), in which case these states are connected with the OR-operator. Only adjacent columns or groups of columns are considered in order not to violate the tree structure principle, which provides an easy way to consult the DT. The limitation to adjacent states stems from the fact that in most cases conditions take on numerical values. The concerning condition states then reveal an order of ranking and contracting non successive states would harm readability. Decision columns corresponding with impossible combinations of condition states are contracted with neighboring decision columns.
A *contracted* DT is a DT in which the number of decision columns is minimized for a given condition order by means of this procedure. Figure 2 shows the contracted DT corresponding to the expanded DT in figure 1.

1. insufficiencies	0		1				more				
2. overall points	-		<half		>=half		<half			>=half	
3. courses passed	-		-		-		<half	>=half		-	
4. attended all exams	Y	N	Y	N	Y	N	-	Y	N	Y	N
1. passed	x	.	.	.	.	.	.	.	.	.	.
2. n passed carry over	.	.	x	.	.	.	.	x	.	x	.
3. not passed	.	x	.	x	.	x	x	.	x	.	x
4. vote	.	.	.	.	x	.	.	.	.	.	.
	1	2	3	4	5	6	7	8	9	10	11

Figure 2: Contracted DT

The minimal number of decision columns to which a DT can be contracted heavily depends on the test order of the conditions. Moreover, this test order can be subject to precedence constraints. A precedence constraint indicates that a certain condition should always be tested before another condition. These constraints can be represented e.g. by means of a precedence matrix $[p_{ij}]$, i.e. a cnum × cnum square matrix of zeroes and ones in which $p_{ij} = 1$ if condition C_i has to precede condition C_j and $p_{ij} = 0$ otherwise. Optimal contraction of a DT involves, besides the contraction process itself, the determination of an optimal condition order, i.e. an acceptable condition order which results in the minimum number of contracted columns. For a table with N conditions, this implies a choice between N! alternative condition

orders (some of which might be infeasible because of the precedence constraints). An *optimal contracted* DT is a contracted DT with a condition order which results in the minimum number of contracted columns. Notice that optimality is defined in terms of minimization of the number of decision columns and not as minimization of the table width (space occupied by the DT). In this way the logical complexity of the decision process is reduced as much as possible (see figure 3). Besides this reduction in logical complexity, optimal contraction also has a positive effect on the time needed to make a decision by means of the table. Therefore optimal contraction of DTs can contribute to an enhanced efficiency and effectiveness in many complex procedural decision situations. The algorithm presented in this paper transforms an expanded DT into an optimal contracted DT.

1. attended all exams	Y					N
2. overall points	<half		>=half			-
3. insufficiencies	-		0	1	more	-
4. courses passed	<half	>=half	-	-	-	-
1. passed	.	.	x	.	.	.
2. n passed carry over	.	x	.	.	x	.
3. not passed	x	.	.	.	.	x
4. vote	.	.	.	x	.	.
	1	2	3	4	5	6

Figure 3: Optimal contracted DT

3. Previous research

The application area of DTs has been limited for a long period (1950-1970) to the world of computer programming. In this era, much research effort was devoted to the (optimal) conversion of DTs into program code.

In the seventies, the application field started to enlarge towards various other domains with logical complexity. More attention was paid to the construction process of the DT, which contributed to the need for computer support for the development of DTs and various automatic transformations and manipulations, among which expansion and consolidation of decision columns.

Algorithms to simplify DTs were presented by Pollack [9], Shwayder [16] and Strunz [17]. However, all of these procedures are either not guaranteeing a minimal solution or not applicable to single hit DTs. In [16] e.g., the DT is transformed into a multiple hit table (i.e. a table with non-exclusive decision columns), which is -from our representational point of view- an inferior variant of the DT [21]. Moreover, in multiple hit tables, the condition order has no influence on the minimum number of columns to which the table can be contracted, while this is precisely a critical issue when working with single hit DTs.

In 1981, Maes proposed an algorithm to find the condition order in a DT which results in the minimum number of contracted decision columns [5]. However, the approach did not deal with groups of decision columns that can be contracted. An improved version was developed by Engelen and Vanthienen [3] [20].

The algorithm in this paper is an extension of this improved version. Various extra features have been added in order to make it applicable in practice (such as the possibility to impose precedence constraints, the possibility to identify impossible condition combinations). Moreover, the algorithm is implemented in the PROLOGA tool, as described in [1].

4. The algorithm

In this section a branch and bound algorithm is presented that determines an optimal condition order of a DT, resulting in the minimum number of contracted decision columns.

4.1. Preliminaries

The theorem on which the algorithm is based, is explained and illustrated with regard to the DT of figure 4[2]. For the sake of conciseness, no specific conditions and actions are given. Conditions and action configurations are represented by a number, while condition states are represented by a letter.

C1	a								b								c							
C2	a				b				a				b				a				b			
C3	a		b		a		b		a		b		a		b		a		b		a		b	
C4	a	b	a	b	a	b	a	b	a	b	a	b	a	b	a	b	a	b	a	b	a	b	a	b
AC	1	1	2	2	3	3	4	4	3	5	6	3	3	4	3	1	3	5	6	3	3	4	3	1

Figure 4: Example DT

Definition
The *action configuration vector* of state s of condition C_k, ACV [k, s], is the vector consisting of the action configurations corresponding with state s of condition C_k from left to right in the expanded DT. The *order* of ACV [k, s], |ACV [k, s]|, is defined as the number of different action configurations in ACV [k, s]. ■

Example
With regard to the DT in figure 4, the following holds:

ACV [1, 1] = [1 1 2 2 3 3 4 4] |ACV [1, 1]| = 4
ACV [1, 2] = [3 5 6 3 3 4 3 1] |ACV [1, 2]| = 5

[2] At this point, it is assumed that no impossible condition combinations occur.

ACV [1, 3] = [3 5 6 3 3 4 3 1] |ACV [1, 3]| = 5
ACV [2, 1] = [1 1 2 2 3 5 6 3 3 5 6 3] |ACV [2, 1]| = 5
ACV [2, 2] = [3 3 4 4 3 4 3 1 3 4 3 1] |ACV [2, 2]| = 3 ■

Theorem
Let C_k be a condition in a DT. A lower bound for the number of decision columns in a contracted DT, in which C_k is the first condition tested, is:

$$LB[k] = \sum_{s=1}^{statnum[k]} \left(|ACV\ [k, s]| - X[k, s] \right)$$

with statnum [k] : the number of condition states of condition C_k;
X [k, s] : the number of action configurations that has to be subtracted in case state s of condition C_k can be contracted with state s-1;
X [k, 1] = 0;
X [k, s] = 0 **if** s > 1 **and** ACV [k, s] ≠ ACV [k, s-1];
X [k, s] = |ACV [k, s]| **if** s > 1 **and** ACV [k, s] = ACV [k, s-1]. ■

Proof
It is a necessary though not sufficient condition for contraction of two decision columns belonging to the same condition state, that these columns have equal action configurations. As a result, the number of decision columns belonging to one condition state, say s, of the condition C_k can be maximally reduced to the number of different action configurations appearing in it (|ACV [k, s]|).
If adjacent states of condition C_k have the same action configuration vector, these condition states can be contracted. In this case, the number of different action configurations belonging to the concerning states, have to be counted only once. ■

Example
With regard to the DT in figure 4, the following holds:

$$\begin{aligned} LB\ [1] &= \left(|ACV[1, 1]| - X[1, 1] \right) + \left(|ACV[1, 2]| - X[1, 2] \right) + \left(|ACV[1, 3]| - X[1, 3] \right) \\ &= (4 - 0) + (5 - 0) + (5 - 5) \\ &= 9 \end{aligned}$$

$$\begin{aligned} LB\ [2] &= \left(|ACV[2, 1]| - X[2, 1] \right) + \left(|ACV[2, 2]| - X[2, 2] \right) \\ &= (5 - 0) + (3 - 0) \\ &= 8 \end{aligned}$$

$$LB\ [3] = 9$$

$$LB\ [4] = 10 \quad ■$$

4.2. The algorithm

The proposed algorithm uses the branch and bound technique. Each node of the search tree corresponds with a partially defined condition order. For each condition

that is not tested yet, the lower bound is calculated for the number of decision columns in a contracted DT which condition order meets the order supplied by the node expanded with the concerning condition. These lower bounds act as limitation factor curtailing the growth of the search tree. Different search strategies are possible (depth first, best first). The search process continues until a complete test order with the smallest lower bound, which then actually equals the number of decision columns in the contracted DT, is found.

The lower bound of a condition in a node of the search tree is determined as follows: with each combination of, contracted if possible, condition states of the conditions that are already fixed, corresponds a subtable of the DT. For each condition (k) that is not fixed yet and for each subtable (t), the minimum number of contracted columns (when this condition is the first condition tested) is calculated as follows (cf. theorem):

$$LB[k, t] = \sum_{s=1}^{statnum[k]} \left(|ACV\ [k, s, t]| - X[k, s, t] \right)$$

with ACV [k, s, t] : ACV [k, s] in subtable t;
X [k, s, t] : X [k, s] in subtable t.

The lower bound of condition k (over all subtables) is then given by:

$$LB[k] = \sum_{t} LB[k, t]$$

Example
Suppose with regard to the DT in figure 4 that condition 2 is already chosen. This condition divides the DT in two subtables: one corresponding with condition state 2a and one corresponding with condition state 2b.

The lower bound of e.g. condition 1 can then be calculated as follows:

$$LB[1] = LB[1, 1] + LB[1, 2] = 10$$

$$LB[1, 1] = \sum_{s=1}^{3} \left(|ACV[1, s, 1]| - X[1, s, 1] \right) = (2 - 0) + (3 - 0) + (3 - 3) = 5$$

$$LB[1, 2] = \sum_{s=1}^{3} \left(|ACV[1, s, 2]| - X[1, s, 2] \right) = (2 - 0) + (3 - 0) + (3 - 3) = 5$$

Suppose now that condition 2 and condition 1 are chosen in this order. These conditions divide the DT in 4 subtables, corresponding with the condition combinations (2a, 1a), (2a, 1b or 1c), (2b, 1a) and (2b, 1b or 1c).

The lower bound of e.g. condition 3 can then be calculated as follows:

$$LB[3] = LB[3, 1] + LB[3, 2] + LB[3, 3] + LB[3, 4]$$
$$= 2 + 4 + 2 + 4$$
$$= 12 \quad \blacksquare$$

The structure of the algorithm when applying the depth first strategy and when retaining one condition order, is given in figure 6. The algorithm operates as follows:
The smallest number of columns found so far is stored in the variable TreeMin. A partial condition order under investigation is extended only if its lower bound is less than the value of TreeMin (in the other case, extension can impossibly result in a better condition order than the one already found). Only extensions that are not conflicting with possible precedence constraints are examined. When a complete condition order is found with a lower bound less than the value of TreeMin, this condition order and its corresponding lower bound become the best solution so far. The algorithm ends when all the partial solutions that are generated are examined in this way.

In PROLOGA, a DT is restricted to have 9 conditions, with a maximum of 6 states for each condition and 1024 decision columns in the expanded DT. Large scale problems are modeled by means of a hierarchy of DTs. These DTs can be transformed separately into optimal contracted DTs. As performance is only related to one table, it does not constitute a problem. Experience has shown that execution time of the algorithm is negligible and produces no noticeable slow-down of an interactive system.

4.3. Illustration

Figure 7 shows the search tree that is generated when the algorithm of figure 6 is applied with respect to the DT in figure 4. The (partially) defined condition order corresponding with a node of the search tree is represented in the first box of each node. The occurrence of e.g. the numbers 1, 4, 0, 0 in this box means that condition 1 is tested first, followed by condition 4, while the test order of condition 2 and condition 3 is not determined yet. The number in the second box of each node concerns the order in which the nodes are generated by the algorithm, while the numbers in the last box represent the lower bounds of the conditions that are not tested yet.

The optimal solution found by the algorithm is the condition order 1, 2, 4, 3. The corresponding contracted DT has 11 decision columns (see figure 5).

C1	a				b or c						
C2	a		b		a				b		
C4	-		-		a		b		a	b	
C3	a	b	a	b	a	b	a	b	-	a	b
AC	1	2	3	4	3	6	5	3	3	4	1

Figure 5: The resulting optimal contracted DT

```
Set the value of TreeMin to infinity;
Set the value of EndOfAlgorithm to false;
Calculate the lower bounds of the elements of the root node, i.e. the
node in which no conditions are fixed yet;
Make the root node current;

While not EndOfAlgorithm do
 Begin
  Select in the current node the next element that has to be
  evaluated;
  Set the value of LB to the lower bound of this element;
  If LB >= TreeMin
  Then
   Begin
      If the element being evaluated is the last of its node
      Then
        Make the first ancestor that still has elements to evaluate
        current, or, if no such ancestor exists, set the value of
        EndOfAlgorithm to true
   End
  Else {LB < TreeMin}
      If the current node is a leaf
      Then {a new minimal condition order is found}
       Begin
        Set the value of TreeMin to the number of decision columns in
        the DT with the condition order of the current node;
        Make the first ancestor that still has elements to evaluate
        current, or, if no such ancestor exists, set the value of
        EndOfAlgorithm to true
       End
      Else {current node is not a leaf}
        If condition order is acceptable (i. e. if all conditions
           that need to precede the current element are already
           chosen)
        Then
         Begin
          Branch the current element;
          Make the child node current
         End
        Else {condition order is not acceptable}
         If the element being evaluated is the last of its node
         Then
          Make the first ancestor that still has elements to evaluate
          current, or, if no such ancestor exists, set the value of
          EndOfAlgorithm to true
End;
```

Figure 6: Structure of the algorithm

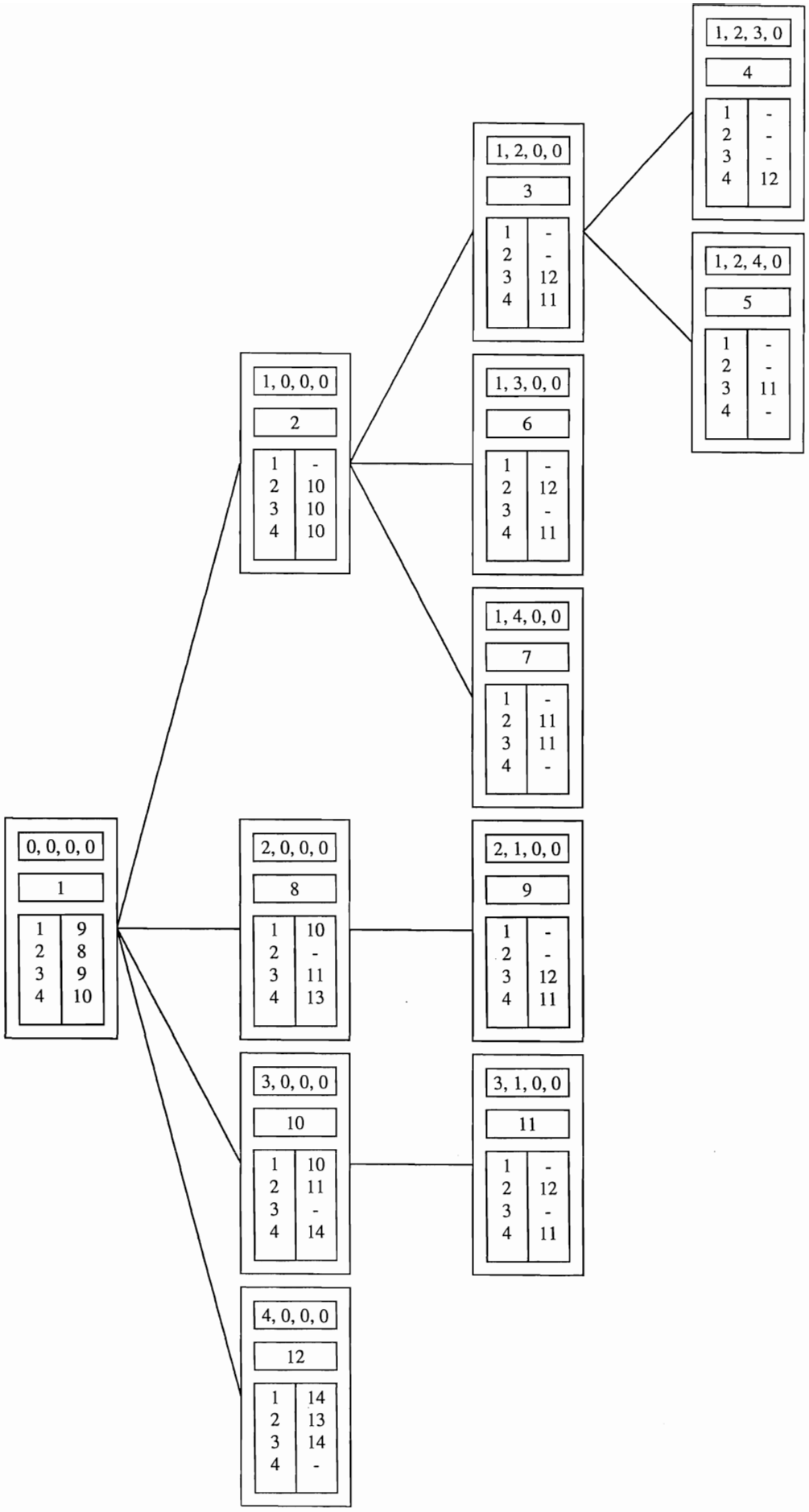

Figure 7: Example of a search tree generated by the algorithm

4.4. *Impossible combinations of condition states*

In what precedes, it was assumed that no impossible combinations of condition states occur in a DT. In practice, however, it is possible that there is no sense in combining certain condition states. Different ways exist to deal with the occurrence of impossibilities in a DT (see [21] for a discussion). One way is to contract impossible combinations of condition states with neighboring decision columns. In this section it is explained how the foregoing algorithm can be extended in order to incorporate this kind of contraction. The difference, which is in the calculation of the lower bound for the number of decision columns in a contracted DT, will be illustrated with regard to the DT in figure 8. In this DT, action configurations are indicated by positive numbers, while the action configuration part corresponding with an impossible combination of condition states is given the number zero.

C1	a						b					
C2	a		b		c		a		b		c	
C3	a	b	a	b	a	b	a	b	a	b	a	b
AC	1	2	1	2	0	1	3	0	0	2	2	0

Figure 8: Example DT with impossible combinations of condition states

Definitions

Let C_k be a condition in a DT and s a condition state of C_k, $s > 1$.

ACV [k, s-1] = [a_1 a_2 ... a_n] and ACV [k, s] = [b_1 b_2 ... b_n] are *contractible* if and only if $\forall i \in \{1, 2, ..., n\}$: if $a_i \neq 0$ and $b_i \neq 0 \Rightarrow a_i = b_i$.

The components c_i of the *contracted vector* of ACV [k, s-1] and ACV [k, s], contracted (ACV [k, s-1], ACV [k, s]), satisfy $c_i = \max(a_i, b_i)$.

The *order* of ACV [k, s], |ACV [k, s]|, is defined as the number of different non-zero action configurations in ACV [k, s]. ■

Example

With regard to the DT in figure 8, the following holds:

ACV [2, 1] = [1 2 3 0] |ACV [2, 1]| = 3
ACV [2, 2] = [1 2 0 2] |ACV [2, 2]| = 2
ACV [2, 3] = [0 1 2 0] |ACV [2, 3]| = 2

ACV [2, 1] and ACV [2, 2] are contractible.
Contracted (ACV [2, 1], ACV [2, 2]) = [1 2 3 2]. ■

The consequence of contracting condition states 2a and 2b is that the impossible combinations of condition states (1b, 2a, 3b) and (1b, 2b, 3a) are related to respectively action configurations 2 and 3 in the contracted DT.

Calculation of the lower bound

In the case that impossible combinations of condition states exist, the calculation of the lower bound for the number of decision columns is more complicated, because contractible action configuration vectors have to be actually contracted. The following algorithm calculates the lower bound for the number of decision columns in a contracted DT in which C_k is the first condition tested:

```
LB [k] := 0;
contr := ACV [k, 1];
for s := 2 to statnum [k] do
 begin
  if contractible (contr, ACV [k, s])
  then contr := contracted (contr, ACV [k, s])
  else
   begin
    LB [k] := LB [k] + |contr|;
    contr := ACV [k, s]
   end;
LB [k] := LB [k] + |contr|;
```

Example

Applying the algorithm with respect to condition 2 of the DT in figure 8 results in the following successive assignments to LB [2]:

$$LB[2] := 0$$

$$LB[2] := LB[2] + |[1\,2\,3\,2]|$$

$$LB[2] := LB[2] + |[0\,1\,2\,0]|$$

The final value for LB [2] is:

$$LB[2] = 5 \quad \blacksquare$$

Remark

It should be noted that applying the above algorithm, not always results in a minimal solution in the case that impossibilities occur. Suppose for instance the following three action configuration vectors, corresponding with the three states of a condition C_k:

[0 2 0 0] [1 0 2 0] [1 3 2 0]

Applying the above algorithm results in a contraction of the first two action configuration vectors and a lower bound LB [k] = 5. However, contraction of the last two action configuration vectors would have resulted in a lower bound LB [k] = 4. This problem could be avoided by checking all possible ways in which the action configuration vectors of a condition can be contracted. This would however result in a severe increase in complexity. Since such a case does not occur frequently in practice and in most cases a reasonable approximation of the minimal solution is found if it occurs, we have chosen to implement the above algorithm in the PROLOGA workbench.

Conclusion

The representational capabilities of the expanded DT make it a valuable tool in knowledge acquisition and verification and validation. The knowledge enclosed in an expanded DT can be represented in several ways. In this paper a branch and bound algorithm is presented to contract DTs in an optimal way. The algorithm outperforms earlier algorithms and procedures with respect to the simplification of DTs. It can be applied in order to optimize the representation of complex decision processes.

References

[1] Bonami, C., *De Optimale Volgorde van Condities in een Beslissingstabel*, Dissertation, K.U.Leuven, Dept. of Applied Economic Sciences, 1993, 80 pp.

[2] CODASYL, *A Modern Appraisal of Decision Tables*, Report of the Decision Table Task Group, ACM, New York, 1982.

[3] Engelen, J., *Minimalisatie van de Omvang van Beslissingstabellen*, Dissertation, K.U.Leuven, Dept. of Applied Economic Sciences, 1985, 89 pp.

[4] Hicks, R. C., Minimizing Maintenance Anomalies in Expert Systems, *Information & Management*, Vol. 28, 1995, pp. 177-184.

[5] Maes, R., *Bijdrage tot een Kritische Herwaardering van de Beslissingstabellentechniek*, Doctoral Dissertation, K.U.Leuven, Dept. of Computer Sciences, 1981, 397 pp.

[6] Maes, R., Van Dijk, J. E. M., On the Role of Ambiguity and Incompleteness in the Design of Decision Tables and Rule-Based Systems, *The Computer Journal*, Vol. 31, No. 6, 1988, pp. 481-489.

[7] Merlevede, P., Vanthienen, J., A Structured Approach to Formalization and Validation of Knowledge, *IEEE/ACM International Conference on Developing and Managing Expert System Programs*, Washington, D. C., Sept. 30 - Oct. 2, 1991, pp. 149-158.

[8] Ngwenyama, O. K., Bryson, N., A Formal Method for Analyzing and Integrating the Rule Sets of Multiple Experts, *Information Systems*, Vol. 17, No. 1, 1992, pp. 1-16.

[9] Pollack, S. L., Hicks, H. T. Jr., Harrison, W. J., *Decision Tables: Theory and Practice*, John Wiley & Sons, Inc., New York, 1971, 179 pp.

[10] Puuronen, S., A Tabular Rule Checking Method, *Proc. Avignon87*, Vol. 1, 1987, pp. 257-268.

[11] Quinlan, J. R., Induction of Decision Trees, *Machine Learning*, Vol. 1, No. 1, 1986, pp. 81-106.

[12] Remus, W. E., An Empirical Investigation of the Impact of Graphical and Tabular Data Presentations on Decision Making, *Management Science*, Vol. 30, No. 5, 1984, pp. 533-542.

[13] Remus, W. E., A Study of Graphical and Tabular Displays and Their Interaction with Environmental Complexity, *Management Science*, Vol. 33, No. 9, 1987, pp. 1200-1205.

[14] Santos-Gomez, L., Darnell, M. J., Empirical Evaluation of Decision Tables for Constructing and Comprehending Expert System Rules, *Knowledge Acquisition*, Vol. 4, 1992, pp. 427-444.

[15] Shiffman, R. N., Greenes, R. A., Use of Augmented Decision Tables to Convert Probabilistic Data into Clinical Algorithms for the Diagnosis of Appendicitis, In: Clayton, P. D., ed., *Symposium on Computer Applications in Medical Care*, Washington D. C., McGraw Hill, 1991, pp. 686-690.

[16] Shwayder, K., Combining Decision Rules in a Decision Table, *Communications of the ACM*, Vol. 18, No. 8, 1975, pp. 476-480.

[17] Strunz, H., *Grundlagen und Anwendungsmöglichkeiten der Entscheidungstabellentechnik bei der Gestaltung rechnergestützter Informationssysteme*, Doctoral Dissertation, University of Köln (West Germany), 1975.

[18] Subramanian, G. H., Nosek, J., Raghunathan, S. P., Kanitkar, S. S., A Comparison of the Decision Table and Tree, *Communications of the ACM*, Vol. 35, No. 1, 1992, pp. 89-94.

[19] Tanaka, M., Aoyama, N., Sugiura, A., Koseki, Y., Integration of Multiple Knowledge Representation for Classification Problems, *Proceedings of the Fifth International Conference on Tools with Artificial Intelligence*, Boston, Mass., Nov. 8-11, 1993, pp. 448-449.

[20] Vanthienen, J., *Automatiseringsaspecten van de Specificatie, Constructie en Manipulatie van Beslissingstabellen*, Doctoral Dissertation, K.U.Leuven, Dept. of Applied Economic Sciences, 1986, 378 pp.

[21] Vanthienen, J., Dries, E., Decision Tables: Refining the Concept and a Proposed Standard, to appear in *Communications of the ACM*.

[22] Vanthienen, J., Dries, E., Illustration of a Decision Table Tool for Specifying and Implementing Knowledge Based Systems, *International Journal on Artificial Intelligence Tools*, Vol. 3, No. 2, 1994, pp. 267-288.

[23] Vanthienen, J., Mues, C., Aerts, A., Wets, G., A Modularization Approach to the Verification of Knowledge Based Systems, *Fourteenth International Joint Conference on Artificial Intelligence* (IJCAI 95), *Workshop on Validation & Verification of Knowledge-Based Systems*, Aug. 19, 1995, Montréal, pp. 96-102.

[24] Vanthienen, J., Wets, G., From Decision Tables to Expert System Shells, *Data & Knowledge Engineering*, Vol. 13, 1994, pp. 265-282.

[25] Vessey, I., Weber, R., Structured Tools and Conditional Logic: An Empirical Investigation, *Communications of the ACM*, Vol. 29, No. 1, 1986, pp. 48-57.

[26] Zhou, X.-J. M., Dillon, T. S., Theoretical and Practical Considerations of Uncertainty and Complexity in Automated Knowledge Acquisition, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 7, No. 5, 1995, pp. 699-712.

Contents