

Temporal Decision Trees or the lazy ECU vindicated *

Luca Console and Claudia Picardi

Dipartimento di Informatica
Università di Torino
Corso Svizzera 185
I-10149, Torino, Italy
{lconsole,picardi}@di.unito.it

Daniele Theseider Dupré

Dip. di Scienze e Tecnologie Avanzate
Università del Piemonte Orientale
Corso Borsalino 54
I-15100, Alessandria, Italy
dtd@mfn.unipmn.it

Abstract

The automatic generation of diagnostic decision trees from qualitative models is a reasonable compromise between the advantages of using a model-based approach in technical domains and the constraints imposed by on-board applications. In this paper we extend the approach to deal with temporal information. We introduce a notion of temporal diagnostic decision tree, in which nodes have a temporal label providing temporal constraints on the observations, and we present an algorithm for compiling such trees from a model-based diagnostic system.

1 Introduction

The adoption of on-board diagnostic software in modern vehicles and similar industrial products is becoming more and more important. This is due to the increasing complexity of systems and subsystems, esp. mechatronic ones like the ABS and fuel injection, and to the increasing demand for availability, safety and easy maintenance. However, in order to keep costs acceptable, esp. for large scale products like cars, limited hardware resources are available in Electronic Control Units (ECUs). Thus, although the model-based approach is very interesting and promising for the automotive domain and similar ones [Console and Dressler, 1999; Sachenbacher *et al.*, 1998], it is still questionable if diagnostic systems can be designed to reason on first-principle models on board. For this reason the following *compilation-based* scheme to the design of on-board diagnostic systems for vehicles was experimented in the Vehicle Model Based Diagnosis (VMBD) BRITE-Euram Project (1996-99), and applied to a Common-rail fuel injection system [Cascio *et al.*, 1999]:

- A model-based diagnostic system, which can be used to solve problems off-line (and then for off-board diagnosis

in the garage), is used to solve a set of significant cases (that is, at least a good sample of the cases that have to be faced on-board). The solution includes the set of candidate diagnoses and, more important, the recovery action(s) that should be performed on-board in that case.

- Cases and the corresponding recovery actions are used to generate a compact on-board diagnostic system. Specifically, in [Cascio *et al.*, 1999] decision trees are generated, using an algorithm based on ID3 [Quinlan, 1986].

However, the system has a major limitation: even if some form of reasoning on the dynamics is used to achieve proper diagnostic results, all abnormal data are considered relative to a single snapshot, and therefore no temporal information is associated with the data in the table used by the compiler and then in the decision tree.

In this paper the decision tree generation approach is extended to take into account temporal information associated with the model, with the manifestations of faults and with the decisions to be made. Using the additional power provided by temporal information requires however new solutions for organizing and compiling the decision trees to be used on board.

The essential idea for generating small decision trees in the temporal case is that *in some cases there is nothing better than waiting*, in order to get a good discrimination, provided that safety and integrity of the physical system is kept into account.

2 Temporal diagnostic decision trees

Defining and compiling decision trees is conceptually simple in the atemporal case. Each node of the tree corresponds to an observable (e.g., a sensor reading) and has several descendants depending on the qualitative values associated with the sensor. The leaves of the tree correspond to actions that can be performed on board.

New opportunities and problems have to be faced in the temporal case. Each fault leads to a different evolution across time, and, for some fault, the appropriate recovery action must be performed within a given time to avoid unacceptable consequences as regards the car integrity and, most importantly, safety.

In the definition of the decision tree, temporal information can be exploited for discriminating faults based on temporal

*This work was partially supported by the EU under the grant GRD-1999-0058, IDD (Integrated Diagnosis and Design), whose partners are: Centro Ricerche Fiat, Daimler-Chrysler, Magneti Marelli, OCC'M, PSA, Renault, Technische Universität München, Université Paris XIII, Università di Torino.

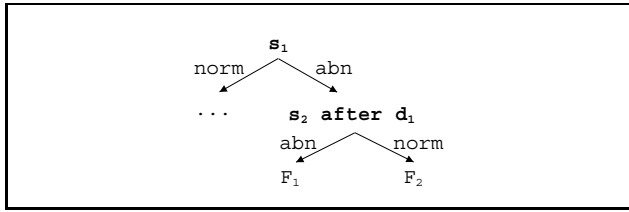


Figure 1: A simple example of temporal decision tree

information, e.g. faults that lead to the same manifestations with different timing. Suppose, for example, that faults F_1 and F_2 both produce an abnormal value for sensor s_1 , but F_1 produces an abnormal value for sensor s_2 after time d_1 while F_2 produces the same abnormal value for s_2 after time $d_2 > d_1$. The two faults cannot be distinguished in an atemporal decision tree and, in general, without temporal information. On the contrary, in the temporal case we can build a tree like the one in figure 1. After observing an abnormal value for s_1 , the system waits for d_1 units of time, and then makes a decision, depending on whether at that time an abnormal value for s_2 is read. However, this behavior is only reasonable in case the recovery actions associated with faults F_1 and F_2 allow the system to wait for d_1 more units of time after fault detection (i.e. in this example, observing an abnormal value for s_1).

Timing information then leads to the definition of *temporal diagnostic decision tree*, i.e. a decision tree such that:

- Each node is labelled with a pair $\langle s, t \rangle$, where s is the name of an observable and t is a time delay.
- The root corresponds to the sensor used for fault detection; in case more than one sensor may be used for such a purpose, there are multiple decision trees, one for each sensor.
- Each edge is labelled with one of the qualitative values of the sensor associated with its starting node.
- Leaves correspond to recovery actions.

The tree is used as follows:

1. Start from the root when a fault is first detected.
2. When moving to a node $\langle s, t \rangle$, wait for t units of time, then read sensor s and move to the descendant of s corresponding to the observed value for the sensor.
3. When moving to a leaf, perform immediately the corresponding recovery action.

In the following sections we present how a temporal diagnostic decision tree can be generated automatically. Before describing that, however, we must discuss the assumptions we make as regards the underlying model-based diagnostic approach.

3 The underlying model-based approach

As regards the temporal dimension, the architecture in [Cascio *et al.*, 1999] relies on either static diagnosis (i.e. diagnosis based on a static model, or based on reasoning in a single state of the system) or a limited form of temporal diagnosis

(see [Brusoni *et al.*, 1998] for a general discussion on temporal diagnosis) where the model of time is purely qualitative (a sequence of qualitative states), abnormal manifestations are assumed to be acquired in a single snapshot t and therefore matched with a single qualitative state, and normality of observations in times earlier than t is used to rule out diagnoses which predict abnormal manifestation that were not detected (see [Panati and Theseider Dupré, 2000]).

What is *not* exploited is the ability of discriminating among explanations for the observations at snapshot t using information acquired at *subsequent* snapshots, which is what we intend to exploit in the present paper, to extend the approach to other forms of temporal models and other notions of temporal diagnosis.

In particular, we consider models of temporal behavior in which temporal information is associated with the relations describing the behavior of a device. We do not make specific assumptions on the model of time, even though, as we shall see in the following, this has an impact on the cases which can be considered for the tree generation. Similarly, we do not make assumptions on the notion of diagnosis being adopted (in the sense of [Brusoni *et al.*, 1998]). This means that the starting point of our approach is a table containing the results of running the model-based diagnostic system on a set of cases, (almost) independently of the model-based diagnostic system used for generating the table.

However, some discussion is necessary on the type of information in the table. In the static case, with finite qualitative domains, the number of possible combinations of observations is finite, and usually small, therefore there are two equivalent ways of building the table:

1. *Simulation approach*: for each fault F , we run the model-based system to predict the observations corresponding to F .
2. *Diagnostic approach*: we run a diagnosis engine on combinations (all relevant combinations) of observations, to compute, the candidate diagnoses for each one of these cases.

In either case, the resulting decision tree contains the same information as the table; if, once sensors are placed in the system, observations have no further cost, the decision tree is just a way to save space and speed up table lookup.

In the temporal case, if the model of time is purely qualitative, a table with temporal information cannot be built by prediction, while it can be built running the diagnosis engine on a set of cases with quantitative information: diagnoses which make qualitative predictions that are inconsistent with the quantitative information can be ruled out. Of course, this cannot in general be done exhaustively, even if observations are assumed to be acquired at discrete times; if it is not, the decision tree generation will be really *learning* from examples.

Thus a prediction approach can only be used in case the temporal constraints in the model are precise enough to generate predictions on the temporal location of the observations (e.g., in case the model included quantitative temporal constraints), while a diagnostic approach can be generate also

in case of weaker (qualitative) temporal constraints in the model.

As regards the observations, we consider the general case where a set of snapshots is available; each snapshot is labelled with the time of observation and reports the value of the sensors (observables) at that time. This makes the approach suitable for different notions of time in the underlying model and on the observations (see again the discussion in [Brusoni *et al.*, 1998]).

We require that some knowledge is available on the recovery actions that can be performed on-board:

- A recovery action a_i is associated with each fault F_i .
- A cost $c(a_i) \in \mathbb{R}$ is associated with each recovery action a_i , due to e.g. the reduction of functionality of the system as a consequence of the action.
- A partial order $\prec$ is defined on recovery actions: $a_i \prec a_j$ in case a_j is stronger than a_i , in the sense that all recovery effects produced by a_i are produced also by a_j . This means that a_j also recovers from F_i although it is stronger than needed. For example, for two recovery actions in the automotive domain [Cascio *et al.*, 1999]: **reduce performance** $\prec$ **limp home** as the latter corresponds to a *very* strong reduction of car performances which simply allows the driver to reach the first workshop. Actions define a lattice, whose bottom corresponds to performing no action and whose top corresponds to the strongest action, e.g., **stop engine** in the automotive domain.
- Costs are increasing wrt $\prec$, that is, if $a_i \prec a_j$ then $c(a_i) < c(a_j)$.
- Actions a_1 and a_2 can be combined into $A = \text{merge}(a_1, a_2)$ defined as follows:

$$A = \begin{cases} \{a_1\} & \text{if } a_2 \prec a_1 \\ \{a_2\} & \text{if } a_1 \prec a_2 \\ \{a_1, a_2\} & \text{otherwise} \end{cases}$$

The merge can be extended to compound actions:

$$\text{merge}(A_1, A_2) = (A_1 \cup A_2) \setminus \{a_i \in (A_1 \cup A_2) \mid \exists a_j \in (A_1 \cup A_2) \text{ s.t. } a_i \prec a_j\}.$$

A simple action can be seen as a special case of compound action, identifying a_i with $\{a_i\}$. In general $A_1, A_2 \preceq \text{merge}(A_1, A_2)$.

- For a compound action A the cost is such that $\max_{a_i \in A} \{c(a_i)\} \leq c(A) \leq \sum_{a_i \in A} c(a_i)$.

A merge of actions must be performed in case of a multiple fault or when alternative preferred (e.g. minimal, or minimum cardinality) diagnoses cannot be discriminated, either because no further sensors are available or there is no time to get further sensor readings.

This actions model is valid only if actions are not fault-dependent, e.g. their cost does not depend on the actual fault(s) in the system. We will discuss this points in section 5.

We now have all the elements to describe the structure of the table in which we collect the results of applying the model-based system on the set of simulated cases:

- The table has one row for each case.
- Each row contains:
 - the set of observations corresponding to the case. In particular, for each observable S_j we have the value in each snapshot (this will be denoted as $S_j^{(i,t)}$ where t identifies the snapshot and i the row of the table);
 - the recovery action to be performed and its cost. This is determined by considering the set of candidate diagnoses for the case under examination and, in case of multiple faults or multiple candidate diagnoses, merging the recovery actions associated with the individual faults occurring in the diagnoses;
 - a probability which corresponds to the prior probability of the candidate diagnoses associated with the row;
 - the maximum number of snapshots that can be used for completing the diagnosis.

This table is the input to the algorithm for generating the temporal diagnostic decision tree.

4 Generating the tree

An important issue to be considered for the generation of decision trees is optimality. A standard decision tree is said to be optimal (with respect to the set of decision trees solving a particular decision problem) if its average depth is minimal. The ID3 algorithm exploits the quantity of information (or its opposite, entropy) carried by an observation as a heuristic to build a decision tree that is, if not optimal, at least good enough.

For *temporal diagnostic decision trees* depth is only a secondary issue, since the ability to select a suitable recovery action in the available time becomes more important. A *suitable* action is one that has the proper recovery effects, but that is not more expensive than necessary. Each time two actions are merged (because of lack of discrimination), we end up with performing an action that could have a higher cost than needed. An expected cost can be associated with a tree.

Definition. Let $L(T)$ denote the set of leaves of a tree T , $R(T)$ the set of candidate rows the tree discriminates, $R(l)$ the set of candidate rows associated with a leaf $l \in L(T)$, $A(l)$ the recovery action for l , and $P(l)$ the probability of l . In particular we have

$$P(l) = \frac{\sum_{r \in R(l)} P(r)}{\sum_{r \in R(T)} P(r)}$$

Then the expected cost $\mathbb{X}(T)$ of T can be defined as:

$$\mathbb{X}(T) = \sum_{l \in L(T)} P(l) c(A(l))$$

Since we are interested in building a tree that minimizes expected cost, some more considerations on costs are worth being made. For any tree T , the set $D(T) = \{R(l) \mid l \in L(T)\}$ is a partition of $R(T)$. It is easy to see that if two trees T_1

and T_2 are such that $D(T_1)$ is a subpartition of $D(T_2)$ then $\mathbb{X}(T_1) \leq \mathbb{X}(T_2)$. Intuitively, T_1 has a higher discriminating power than T_2 , and therefore it cannot have a higher expected cost. It follows that if we build the tree with highest discriminating power we automatically build a tree with minimum expected cost. However, *maximum discriminating power* is a relative, concept: it ultimately depends on the data available in the table from which the tree is built. Let us denote with $\mathbb{I}$ the set of all pairs $\langle \text{observable}, \text{time} \rangle$ in the initial table. To fully exploit the discriminating power in $\mathbb{I}$, a tree could be based on *all* the pairs in it. The partition induced on the set of candidates by such a tree is a subpartition of that induced by any other tree that makes use of pairs in $\mathbb{I}$. Therefore the cost of this tree is the *minimum achievable expected cost*.

Unfortunately this is not the tree we want to build, since as well as minimum expected cost it has maximum depth. The algorithm we propose builds a tree with minimum expected cost while trying to minimize its depth, using the criterion of minimum entropy. The algorithm is essentially recursive. Each recursive call is composed of the following steps: (i) it takes as input a set of candidates, represented by table rows; (ii) it selects a pair $\langle \text{observable}, \text{time} \rangle$ for the current tree node; (iii) it partitions the set of rows accordingly to the possible values of the selected attribute; (iv) it generates a child of the current node for each block of the partition; (v) it performs a recursive call on each child.

The difficult step is (ii). First of all, let us notice that, differently from what happens in ID3, the set of pairs from which we can choose is not the initial set $\mathbb{I}$, nor it is the same in all recursive calls. The set of *available pairs* depends on the current snapshot - a tree node cannot consider a snapshot previous to that of its parent - and on the deadline, which in turns depends on the set of candidates. Choosing a pair with a time delay greater than 0 for the current node means making all snapshots between the current one and the selected one unavailable for all the descendants of this node. A pair may be discarded that was necessary in order to attain the minimum expected cost. Of course, the higher the time delay, the higher the risk.

Step (ii) can then be divided in two substeps: first, determine the maximum time delay that does not compromise the minimum expected cost; second, select the pair with lowest entropy between those with a safe delay. In order to determine if a particular delay is safe, we must associate with each possible delay t the minimum expected cost $\chi(t)$ that it is possible to obtain *after* selecting it. Then, since a 0-delay is safe by definition, the delays t with $\chi(t) = \chi(0)$ can be declared safe.

The easiest way to compute $\chi(t)$ is to build the tree that considers *all* the pairs still available after t , as in the definition of minimum achievable expected cost. Since we only need the leaves, and not the whole tree, we can simply consider the partition induced on the set of candidates by the combination of all available pairs. We will refer to this partition as the *global partition for t* .

Doing this for all delays can be computationally expensive; luckily there is a more efficient way to obtain the same result. If we consider two possible delays t_1, t_2 with $t_1 < t_2$ we have that the set of available pairs for t_2 is a subset of that for t_1 .

This means that, if we compute first the global partition for t_2 , we can build the global partition for t_1 starting from that of t_2 and considering only the extra pairs available for t_1 . If we apply this *backward strategy* to the set of all possible delays, we immediately see that we can compute $\chi(t)$ for all t *while* computing $\chi(0)$. In this way we will be able to determine which delays are safe and which are not looking only *once* at the set of pairs that are initially available. If we denote by N the number of rows in the table, by S the number of sensors, by T the number of snapshots, and by k the number of values that each sensor can assume, we have that in the worst case this algorithm behaves as $O(\frac{N^2 \cdot T \cdot S}{k})$. The algorithm as presented in this paper is an improvement, in terms of efficiency, of the original version we described in [Console *et al.*, 2000].

4.1 The algorithm

The algorithm for compiling the tree is reported in figure 2. To start the algorithm up, the sensors used for fault detection must be determined, since a separate tree will be built for each of them. Notice that each tree covers a subset of cases, depending on which abnormal measurement activates diagnosis.

Given the sensor used as the root of the tree, the values the sensor can assume induce a partition on the set of cases, and a subtree must be generated for each block of the partition. This is done in the TEMPORALID3 function, that takes in input a set of cases $\mathcal{R}$ and returns a temporal decision tree along with its expected cost.

In order to express delays instead of absolute times, TEMPORALID3 needs a notion of *current snapshot* that varies from one call to another, and that corresponds to the last snapshot considered. More precisely, since each call to TEMPORALID3 builds a subtree, the current snapshot is the one in which the parent node's observation takes place. However, since the delay between fault occurrence and fault detection depends on which fault is supposed to occur, each candidate $r \in \mathcal{R}$ has to maintain a pointer to its own *current snapshot*; we denote this pointer by $\tau(r)$. At the beginning, for each r , $\tau(r)$ is set to the snapshot at which the fault (or combination of them) described by r is detected.

Moreover, we denote by $P(r)$ the prior probability of r , by $T(r)$ its deadline for a recovery action, and by $A(r)$ its recovery action (either simple or compound).

Let us briefly describe TEMPORALID3. First, it considers all those situations in which it should simply return a tree leaf. This happens when: (i) the input set does not need further discrimination, that is, all the cases in $\mathcal{R}$ have the same recovery action; or (ii) in the time left for selecting an action there are no more discriminating pairs (a particular case is when there is no time left at all). This means that it is not possible to distinguish between the candidates in $\mathcal{R}$ and therefore the selected recovery action is the merge of $\{A(r) \mid r \in \mathcal{R}\}$.

If none of the previous cases holds, TEMPORALID3 uses the *backward strategy* described earlier in order to compute, for each delay t , the expected cost $\chi(t)$, thus finding out which delays are safe. It exploits three subfunctions: PAIRS, SPLIT and EXPECTEDCOST, which, for the sake of conciseness, we describe without providing their pseudocode.

```

function TEMPORALID3( $\mathcal{R}$ )
  returns a pair ( $Tree$ ,  $Cost$ )
begin
  if  $\exists A : \forall r \in \mathcal{R} (A(r) = A)$  then
    {all candidates in  $\mathcal{R}$  have the same recovery action}
    return ( $Tree = \langle A \rangle$ ,  $Cost = c(A)$ );
   $T \leftarrow \min\{T(r) - \tau(r) \mid r \in \mathcal{R}\}$ ; {current deadline}
   $A \leftarrow \text{merge}\{A(r) \mid r \in \mathcal{R}\}$ ;
  if  $T < 0$  then {no time left}
    return ( $Tree = \langle A \rangle$ ,  $Cost = c(A)$ );
  {Backward strategy:}
   $\tilde{t} \leftarrow \text{undefined}$ ;
   $\chi \leftarrow \text{undefined}$ ;
   $\Pi \leftarrow \{\mathcal{R}\}$ ;
  for each  $t$  in  $\{T, T-1, \dots, 0\}$  do begin
     $\mathbb{P} \leftarrow \text{PAIRS}(t, t)$ ;
    if  $\mathbb{P} \neq \emptyset$  then begin
      do begin
         $\Pi \leftarrow \text{SPLIT}(\Pi, \mathbb{P})$ ;
        for each  $\pi \in \Pi$  do begin
           $T(\pi) \leftarrow \min\{T(r) - \tau(r) \mid r \in \pi\}$ ;
           $\mathbb{P} \leftarrow \text{PAIRS}(T+1, T(\pi))$ ;
        end
        while  $\mathbb{P} \neq \emptyset$ ;
         $\chi_{\text{new}} \leftarrow \text{EXPECTEDCOST}(\Pi)$ ;
        if ( $\chi$  is undefined) or ( $\chi_{\text{new}} < \chi$ ) then begin
           $\chi \leftarrow \chi_{\text{new}}$ ;
           $\tilde{t} \leftarrow t$ ;
        end
      end
    if  $\tilde{t}$  is undefined then
      {no more discriminating pairs}
      return ( $Tree = \langle A \rangle$ ,  $Cost = c(A)$ );
     $\langle o_{\text{ok}}, t_{\text{ok}} \rangle \leftarrow$  the pair with minimum entropy
      between those with  $0 \leq t \leq \tilde{t}$ ;
    for each  $r \in \mathcal{R}$  do {Update the value
      of the current snapshot for each candidate}
       $t(r) \leftarrow t(r) + t_{\text{ok}}$ ;
     $\Pi \leftarrow \text{SPLIT}(\{\mathcal{R}\}, \{\langle o_{\text{ok}}, t_{\text{ok}} \rangle\})$ ;
    for each  $\pi \in \Pi$  do
      {Build the subtrees}
      ( $\text{Subtree}(\pi), \mathbb{X}(\pi)$ )  $\leftarrow$  TEMPORALID3( $\pi_i$ );
    return ( $Tree = \langle \langle o_{\text{ok}}, t_{\text{ok}} \rangle, \{\text{Subtree}(\pi) \mid \pi \in \Pi\} \rangle$ ,
       $Cost = \sum_{\pi \in \Pi} \frac{\sum_{r \in \pi} P(r)}{\sum_{r \in \mathcal{R}} P(r)} \cdot \mathbb{X}(\pi)$ );
  end.

```

Figure 2: The function TEMPORALID3

PAIRS returns all the pairs with a delay between the specified values.

SPLIT splits an existing partition using the set of pairs in input. A partition block can be split by a pair only if the pair can provide a value for all the candidates in the block. This is important since some pairs are beyond the deadline and thus do not provide a value for all the candidates.

Row	S_1					S_2					S_3					Act	T_i	P_i	Cost			
	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5				
R_1	n	n	n	n	v		n	l	v	v	v		n	n	n	n	n		a_1	4	$\frac{1}{4}$	100
R_2	n	n	n	n	v	v	n	l	l	l	v	v	n	n	n	n	n	v	a_2	5	$\frac{1}{4}$	10
R_3	n	n	n	n	h	h	n	l	l	l	l	v	n	n	n	n	n	h	a_3	5	$\frac{1}{4}$	10
R_4	n	n	n	n	v		n	l	v	z	z		n	n	n	n	n		a_4	4	$\frac{1}{4}$	2

Table 1: A sample table of data

EXPECTEDCOST computes the expected cost associated with a partition. The partition can be regarded as a set of tree leaves, and therefore it is possible to apply the definition of expected cost.

When we compute $\chi(t)$, as a result of the *backward strategy* we can exploit the partition we created for a delay of $t+1$. First of all we consider the “new” pairs that are available at snapshot t , that is, those pairs returned by $\text{PAIRS}(t, t)$. However it may be that for some blocks of the newly created partition there are some more pairs available, because these smaller blocks have a looser deadline than the older ones. Therefore we keep on (i) creating the new partition; (ii) computing the new deadline for each block; (iii) adding the new pairs due to the new deadline, if any. This is repeated until there are no more pairs to add.

After determining the maximum safe delay, denoted by $\tilde{t}$, the algorithm selects the pair with minimum entropy among those with a delay t such that $0 \leq t \leq \tilde{t}$. The selected pair is denoted by $\langle o_{\text{ok}}, t_{\text{ok}} \rangle$.

Finally, TEMPORALID3 updates the current snapshot pointers, partitions $\mathcal{R}$ according to the selected pair, and calls recursively TEMPORALID3 on each block of the partition.

4.2 An example

Let us consider a simple example. There are three sensors, S_1 , S_2 and S_3 ; each one of them can have five possible values: normal (n), high (h), low (l), very low (v) and zero (z). Possible recovery actions are denoted by a_1, a_2, a_3, a_4 , with the following partial order: $a_4 \prec a_2 \prec a_1, a_4 \prec a_3 \prec a_1$. The only compound action that can arise from a merge operation is $\{a_2, a_3\}$ and we assume its cost is $c(a_2) + c(a_3) = 20$. We assume that the faults associated with the four rows have the same prior probability. Initial data are shown in table 1.

We will build the tree whose root is sensor S_2 ; this is the only tree to be built since each fault in the table is first detected on S_2 . If, for example, cases R_1 and R_2 were first detected on S_2 while R_3 and R_4 on S_1 , we would have had to build two separate trees, the first with root S_2 dealing with R_1, R_2 and the second with root S_1 dealing with R_3, R_4 .

The root of the tree is thus $\langle S_2 \rangle$ but it does not partition the set of candidates, so we still have $\mathcal{R} = \{R_1, R_2, R_3, R_4\}$, and $\tau(R_i) = 1$ for $i = 1, 2, 3, 4$. In this example all faults show themselves with the same delay wrt fault occurrence; this is the reason why all $\tau(R_i)$ s have the same value. In a different situation each would point to the first snapshot for which sensor S_2 has a non-normal reading. Diagnosis must

be completed by snapshots 4, corresponding to a delay of 3 snapshots. Using the backward strategy we first find out that with a delay of 3 we would not be able to discriminate R_1 and R_2 , and therefore $\chi(3) = 53$. On the other hand, if we consider a delay of 2 we are able to discriminate all faults, thus $\chi(0) = \chi(1) = \chi(2) = 30.5$. The minimum entropy criterion prompts us to choose the pair $\langle S_2, 2 \rangle$, and we obtain the partition $\mathcal{R}_1 = \{R_1\}, \mathcal{R}_2 = \{R_2, R_3\}, \mathcal{R}_3 = \{R_4\}$. For $\mathcal{R}_1$ and $\mathcal{R}_3$ the diagnosis is completed and the algorithm returns respectively $\langle a_1, 100 \rangle$ and $\langle a_4, 2 \rangle$. For $\mathcal{R}_2$ we have that the deadline is moved, and we can choose between delays 0, 1 and 2. However it is immediately clear that $\chi(0) = \chi(1) = \chi(2) = 10$; moreover all available pairs have the same entropy (quite obvious since there are only two candidates left). We can choose for example $\langle S_2, 1 \rangle$ and the recursive calls return $\langle a_2, 10 \rangle$ and $\langle a_3, 10 \rangle$.

The resulting decision tree is shown in figure 3.

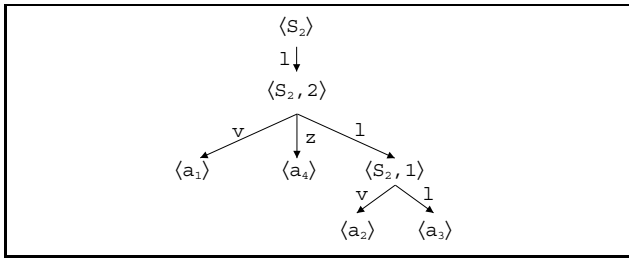


Figure 3: Resulting decision tree

5 Conclusions

In this paper we introduced a new notion of diagnostic decision tree that takes into account temporal information on the observations and temporal constraints on the recovery actions to be performed. In this way we can take advantage of the discriminatory power that is available in the model of a dynamic system. We presented an algorithm that generates temporal diagnostic decision trees from a set of examples, discussing also how an optimal tree can be generated.

The automatic compilation of decision trees seems to be a promising approach for reconciling the advantages of model-based reasoning and the constraints imposed by the on-board hardware and software environment. It is worth noting that this is not only true in the automotive domain and indeed the idea of compiling diagnostic rules from a model has been investigated also in other approaches (see e.g., [Darwiche, 1999; Dvorak and Kuipers, 1989]). [Darwiche, 1999], in particular, discusses how rules can be generated for those platforms where constrained resources do not allow a direct use of a model-based diagnostic system.

Future work on this topic is mainly related to recovery actions. While developing the algorithm, we assumed to have a model of actions and costs where actions are not fault-dependent (see section 3). Modeling such a situation would mean to introduce (a) a more sophisticated cost model where the cost of an action can depend on the behavioral mode of some system components, and (b) a more detailed deadlines

model, that expresses the cost of not meeting a deadline. Notice that this does not affect the core part of the algorithm, i.e. the one that computes the expected costs related to different choices. However having fault-dependent costs requires a different criterion for stopping the discrimination process and building a tree leaf, since meeting the deadline becomes a matter of costs and benefits rather than a strict requirement.

Some other generalizations are more challenging, since they affect also the core part of the algorithm. As an example, we are currently investigating how to integrate actions and candidate discrimination, e.g. including probing actions (which modify the system state) and sequences of actions interleaved with measurements.

References

- [Brusoni *et al.*, 1998] V. Brusoni, L. Console, P. Terenziani, and D. Theseider Dupré. A spectrum of definitions for temporal model-based diagnosis. *Artificial Intelligence*, 102(1):39–79, 1998.
- [Cascio *et al.*, 1999] F. Cascio, L. Console, M. Guagliumi, M. Osella, A. Panati, S. Sottano, and D. Theseider Dupré. Generating on-board diagnostics of dynamic automotive systems based on qualitative deviations. *AI Communications*, 12(1):33–44, 1999. Also in Proc. 10th Int. Workshop on Principles of Diagnosis, Loch Awe, Scotland.
- [Console and Dressler, 1999] L. Console and O. Dressler. Model-based diagnosis in the real world: lessons learned and challenges remaining. In *Proc. 16th IJCAI*, pages 1393–1400, Stockholm, 1999.
- [Console *et al.*, 2000] L. Console, C. Picardi, and D. Theseider Dupré. Generating temporal decision trees for diagnosing dynamic systems. In *Proc. DX 00, 11th Int. Workshop on Principles of Diagnosis*, Morelia, 2000.
- [Darwiche, 1999] A. Darwiche. On compiling system descriptions into diagnostic rules. In *Proc. 10th Int. Work. on Principles of Diagnosis*, pages 59–67, 1999.
- [Dvorak and Kuipers, 1989] D. Dvorak and B. Kuipers. Model-based monitoring of dynamic systems. In *Proc. 11th IJCAI*, pages 1238–1243, Detroit, 1989.
- [Panati and Theseider Dupré, 2000] A. Panati and D. Theseider Dupré. State-based vs simulation-based diagnosis of dynamic systems. In *Proc. ECAI 2000*, 2000.
- [Quinlan, 1986] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.
- [Sachenbacher *et al.*, 1998] M. Sachenbacher, A. Malik, and P. Struss. From electrics to emissions: experiences in applying model-based diagnosis to real problems in real cars. In *Proc. 9th Int. Work. on Principles of Diagnosis*, pages 246–253, 1998.