ELSEVIER

Artificial Intelligence 88 (1996) 317-347

Artificial Intelligence

Using action-based hierarchies for real-time diagnosis

David Ash *, Barbara Hayes-Roth

Knowledge Systems Laboratory, Computer Science Department, Gates Computer Science Building, Stanford University, Stanford, CA 94305, USA

Received February 1996; revised March 1996

Abstract

An intelligent agent diagnoses perceived problems so that it can respond to them appropriately. Basically, the agent performs a series of tests whose results discriminate among competing hypotheses. Given a specific diagnosis, the agent performs the associated action. Using the traditional information-theoretic heuristic to order diagnostic tests in a decision tree, the agent can maximize the information obtained from each successive test and thereby minimize the average time (number of tests) required to complete a diagnosis and perform the appropriate action. However, in real-time domains, even the optimal sequence of tests cannot always be performed in the time available. Nonetheless, the agent must respond. For agents operating in real-time domains, we propose an alternative action-based approach in which: (a) each node in the diagnosis tree is augmented to include an ordered set of actions, each of which has positive utility for all of its children in the tree; and (b) the tree is structured to maximize the expected utility of the action available at each node. Upon perceiving a problem, the agent works its way through the tree, performing tests that discriminate among successively smaller subsets of potential faults. When a deadline occurs, the agent performs the best available action associated with the most specific node it has reached so far. Although the action-based approach does not minimize the time required to complete a specific diagnosis, it provides positive utility responses, with step-wise improvements in expected utility, throughout the diagnosis process. We present theoretical and empirical results contrasting the advantages and disadvantages of the information-theoretic and action-based approaches.

Keywords: Reactive planning; Decision trees; Diagnosis; Real-time planning; Heuristics

* Corresponding author. Present address: Brightware Inc., 90 Park Avenue, Suite 1600, New York, NY 10016, USA. E-mail: 70363.45@compuserve.com.

PII S0004-3702(96)00024-0

Table 1
Desiderata for a parachutist on perceiving a delay in parachute opening

Possible situations	Conditional probabilities	Optimal actions	Deadline
Good parachute	0.1	Steer normally	Now + 120 seconds
Low speed malfunction	0.4	If malfunction on deadline: Release main parachute Activate reserve parachute	Now + 45 seconds
High speed malfunction	0.4	If malfunction on deadline: Release main parachute Activate reserve parachute	Now + 5 seconds
No parachute	0.1	Activate reserve parachute	Now + 3 seconds

1. Introduction

Consider a parachutist in free fall whose parachute seems to be taking longer than usual to open. In order to take appropriate action, the parachutist must discriminate among four qualitatively different situations: a good parachute that will open soon, a malfunctioning parachute with a low-speed fall, a malfunctioning parachute with a high-speed fall, or no parachute at all. As summarized in Table 1, these situations have different conditional probabilities (given the initial observation), different optimal responses, and different deadlines for response. Two diagnostic tests are available to the parachutist: (a) look at the altimeter for a few seconds to classify the situation as low speed or high speed; and (b) look at the main parachute to classify the situation as one involving a good parachute, a malfunctioning parachute, or no parachute at all. Which test should the parachutist perform first?

From an information-theoretic perspective, the parachutist should look at the altimeter first, because this test will distinguish between the two most likely hypotheses—low-speed versus high-speed malfunctions. However, the outcome of this test does not identify an appropriate response. If the rate of fall is low, there is either a good parachute or a low-speed malfunction. Should the parachutist steer normally or release the main parachute and then activate the reserve? If the rate of fall is fast, there is either a high-speed malfunction or no parachute at all. Should the parachutist release the main parachute and then activate the reserve or simply activate the reserve? Regardless of the outcome of the altimeter test, the parachutist cannot choose an appropriate action until he performs the second test, looking at the parachute. The real problem is that there may not be enough time to perform that test before the deadline occurs.

From a utility-oriented perspective, the parachutist should look at the parachute first because the results of this test will identify positive utility actions, particularly for the most critical contingencies. If there is a good parachute, the parachutist should steer normally. If there is no parachute at all, he should activate the reserve immediately. If there is a parachute malfunction, even without knowing whether he is falling at low speed or high speed, the parachutist has an appropriate response, but need not perform it until the deadline occurs. If the parachute malfunction persists until the deadline, the parachutist should release the main parachute and then activate the reserve; otherwise if the main parachute should inflate before the deadline, the parachutist should use it

and keep his reserve parachute in case of a subsequent problem. The default deadline in this case should be the shorter of the two deadlines for the competing fault hypotheses. In the case of the above table which assumes an original opening altitude of about 2500 feet, this would be Now + 5 seconds, but with higher opening altitudes more time would be available. During this interval, the parachutist can start to perform the second test, looking at the altimeter. If he determines that he has a low-speed malfunction, the parachutist can extend the deadline to Now + 45 seconds, increasing the chance that he can use his main parachute and keep his reserve. In sum, by performing the parachute test first, the parachutist can choose an appropriate action immediately; in some cases, if time permits, the parachutist can improve his choice of action by subsequently performing the altimeter test.

This simple example of the parachutist illustrates key features of real-time diagnosis problems:

Perceived problems require some form of response by an uncertain, possibly fault-dependent deadline. For example, no parachute at all requires an almost immediate response when the original opening occurred at 2500 feet, while a parachute malfunction allows a variable latency, depending on the speed of falling as well as the altitude at which the problem is perceived.

Different tests discriminate among different subsets of potential faults. For example, the altimeter test discriminates between two subsets of faults (good parachute or low-speed malfunction versus no parachute or high-speed malfunction), while the parachute test discriminates between three subsets of faults (good parachute, malfunctioning [high speed or low speed] parachute, and no parachute).

Different tests require different amounts of time and other resources. For example, both the altimeter test and the parachute test require use of the parachutist's eyes, while the altimeter test takes a few seconds longer to perform than the parachute test, primarily because the altimeter must be observed over a period of several seconds to accurately estimate rate of fall.

Sometimes the sequence of tests required for specific diagnosis of a perceived problem cannot be completed by the deadline. For example, if the parachutist first perceives his problem at a lower altitude, he may not have enough time to perform both the altimeter and parachute tests.

Different responses have different utilities for different faults. For example, immediately activating the reserve parachute has high utility in the case of no parachute at all, lower utility in the case of a malfunctioning parachute (because of possible entanglement with the malfunctioning main parachute), and very low (actually negative) utility in the case of a good parachute.

Specific diagnoses permit optimal (within the available knowledge) responses. For example, knowing that he has a malfunctioning parachute at low speed (as opposed to a malfunction at high speed) allows the parachutist to wait longer for his main parachute to inflate before releasing it and activating his reserve parachute.

Nonspecific diagnoses permit positive utility nonspecific responses. For example, if the parachutist knows only that he has a malfunctiong parachute, but does not know whether it is at low speed or high speed, he can release his main parachute and activate his reserve parachute for a positive utility in either case.

The agent has sufficient advance time, knowledge, and other resources to prepare a context-appropriate decision tree for diagnosis and action. For example, the parachutist can anticipate that his main parachute might not open on time, go through the kind of analysis described above, and decide that he will respond to such situations by first performing the parachute test and, if time and circumstances warrant, then performing the altimeter test.

In this paper, we present an action-based approach to diagnosis, in which: (a) each node in the diagnosis decision tree is augmented to include an ordered set of actions, each of which has positive utility for all of its children in the tree; and (b) the tree is structured to maximize the expected utility of the action available at each node. Upon perceiving a problem, the agent works its way through the tree, performing tests that discriminate among successively smaller subsets of potential faults. When a deadline occurs, the agent performs the best available action associated with the most specific diagnosis node it has reached so far.

The remainder of this paper is organized as follows. In Section 2, we formally characterize requirements for the two phases of real-time diagnosis, reactive planning and reactive plan execution. In Section 3, we describe our action-based approach. Sections 4–6 evaluate the action-based approach using formal analysis, abstract experiments, and demonstration in a particular domain, intensive care monitoring. Sections 7 and 8 summarize conclusions and identify open problems for future research.

2. The problem

At the most basic level, we are interested in the problem of identifying and acting on one of a set of *faults*. For the purposes of this paper, we define a fault as the most specific diagnosis in which an agent is interested within a given domain. Thus, a fault is a property both of the domain and the goals of an agent. For example, in the parachuting domain, there is actually more than one different type of total malfunction—the parachute might remain stuck in its container, or the parachute may have been incompetently packed, etc. The difference between these total malfunctions is completely irrelevant to the parachutist in free fall, and hence from the point of view of the parachutist faced with a diagnosis problem, they are one and the same fault. At a later point, when debriefing the malfunction on the ground, the two different types of total malfunction might be viewed as different faults because the goal then is not critical response but rather an attempt to prevent a reoccurrence of the malfunction.

For dealing with a set of faults, the agent will have available to it a *reactive plan*. Later in this paper, we will analyze in detail a particular type of reactive plan—an augmented form of decision tree called an action-based hierarchy—but for now it is most important to realize that a reactive plan consists of a series of actions that an agent may take in order to diagnose a particular fault in real time, together with actions for remedying the fault that it diagnoses. Because the reactive plan is intended to be deployed in real time, it is desirable that it consume a minimum of computational time and other resources, and that it have an anytime flavor [14] to it which will allow it to identify useful actions to perform even in cases when resources do not permit its running to completion.

For the purposes of this paper, the *reactive planner* that constructs the reactive plan may be quite distinct from the agent that executes a reactive plan. Consider, for example, a pilot faced with making an emergency landing because of engine failure. The pilot has a checklist—a reactive plan—for dealing with this emergency situation. However, it is generally not the case that the pilot designed the checklist. Although the pilot is the agent who will execute the plan, the checklist is likely to be recommended for all pilots flying a particular type of aircraft. This distinction is important because it indicates differences in the desired properties of the reactive plan versus the reactive planner. The reactive planner need not be especially computationally efficient, because it may have substantial time to complete a plan prior to it ever being needed in real time. However, the plan that it generates must be computationally efficient for the reasons noted above. This distinction will guide the design of reactive plans in this paper.

Partial diagnoses are quite possible in many domains, and it is essential that the reactive plan be able to handle the possibility of making a partial diagnosis. For example, a physician may recommend a transfusion if he/she knows a patient's blood type and that the patient has an anemic condition—even without knowing the specific type of anemic condition the patient is suffering from. Such a recommendation is not necessarily optimal—knowing the particular type of anemia would enable an optimal recommendation—but may be necessary to save the life of the patient. The reactive plan must enable an agent to reach similar goals—finding an optimal acton if time permits, but taking an action that is less than optimal if necessary because of a deadline.

We take the view that a partial diagnosis is a set of faults, together with their current probabilities, and a complete diagnosis is a single fault, with an attached probability of 1.0. Clearly this represents an approximation to the real situation, since we usually cannot assert anything with probability 1.0, and it would certainly be an interesting topic for further research to provide more sophisticated reasoning under uncertainty. If we went this route, we would need a mechanism for ruling a fault out even though its probability is not 0.0. As the reactive plan is executed, the diagnosis will gradually be refined from the set of all faults in the domain, with their attached prior probabilities, to the particular fault that is present in a given situation.

We have stated that we view a fault as something that needs to be corrected. Because of this, the agent also has a set of *corrective actions*, sometimes also known as *responses*, which it can take in executing the reactive plan. Because the agent is executing a *reactive* plan, it is not interested in designing a long sequene of corrective actions: rather it views the execution of a single corrective action as being a solution to a fault. However, the reactive planner recognizes that some solutions are potentially better than others, and so it has a notion of the *value* of performing a corrective action. This is a purely heuristic estimate given either directly by the domain expert or by some well-understood computation within the domain. For the moment, it is assumed to be positive—negative "values" (actually costs of actions) will be discussed below. For each combination of a corrective action and a fault, there is a value number representing the value of that action for that fault. The resulting matrix is expected to be rather sparse in that most corrective actions have no value for most faults. For example, filling the tires with air will not fix a broken-down engine, but the reactive planner needs to know this either explicitly or implicitly.

Because of the sparseness of the value matrix, it is not expected that a domain expert would need to fill in the entire matrix. Rather, the domain expert would fill in values only for those actions that have an interesting effect on particular faults, and the rest of the matrix would be automatically filled in with zeroes. Indeed, in a large domain, the entire matrix would not need to be represented explicitly even internally within the reactive planner.

There are also *costs* associated with pairs of actions and faults. Again, the cost is a heuristic estimate of the cost of performing a particular action in the presence of a particular fault. Here we expect that in most cases the cost of performing an action will be independent of the particular fault it is being applied to. This captures the notion that if the action has no value for the particular fault, then the cost of performing it makes it undesirable to perform that action. Because the cost of performing an action is in general independent of the fault involved, again the cost matrix can be provided by the expert by simply providing costs for the actions and then noting exceptions for particular faults.

In practice, we combine the notion of value and cost into a single notion of total *utility*, which represents the difference between the value and the cost. This utility can therefore be negative. The difference between value and cost appears to be the appropriate combining function. If we took the ratio, actions with little value but infinitesimal cost would be more desirable than actions with high value but moderate cost—this is clearly not a desirable situation.

The utility of the agent's response declines over time. For example, in the parachuting domain, there is a very sharp decline in the utility of the response as the parachutist gets closer to the ground. Note that this decline does not occur instantaneously, for a couple of reasons: the time is measured in seconds, but the exact amount of time that it will take to reach the ground is not known in advance; and also, there is a time range, very close to the ground, where a response, though it may save the parachutist's life, is likely to still result in serious injury. The decline in value of the utility of response plays a major role in the *design* of the reactive plan, because in designing a reactive plan, the planner will try to make sure that the best possible response is available to the agent before its utility decays to the point of being essentially useless.

The time axis in these utility graphs is measured from the time when the reactive plan is first invoked. The agent will generally invoke the reactive plan after noticing that something appears to be going wrong—perhaps it had a regular plan which called for a parameter value to be in a particular range, and the value is outside that range. It is obvious that the invocation of the reactive plan does not necessarily correspond to the time when the fault first appeared, so the reader may ask why we do not measure time from the time when the fault appeared. The answer is that the reactive planner cannot make use of such knowledge. If it knows approximately how long the agent will have after invoking the reactive plan to take action, it can construct the plan accordingly. But if the planner only knows how long the agent will have since the fault first appeared, it cannot be sure when the plan will be invoked and therefore will be operating in the dark. Thus, in providing utility decay information to the reactive planner, one must be careful to measure time from the point where the reactive plan will be invoked—which might be when the first indication of the fault appeared, not when the fault itself appeared.

In order to help the agent diagnose one of the faults, it has available to it a set of *tests* which it can perform. Each test has a set of possible outcomes, and each outcome is a set of faults (a subset of the full set). Thus, when a test is performed and an outcome is obtained, the agent knows that the actual fault belongs to the set corresponding to that outcome. If the agent already has information from another test or group of tests that the actual fault falls within some *differential diagnosis* (with attached probabilities), then the agent can take the intersection of the differential diagnosis with the current outcome to produce a new, smaller differential diagnosis. Gradually by narrowing down the differential the agent hopes to reduce it to a single fault and complete the diagnosis process. The fault sets associated with the outcomes for a given test need not be disjoint—some tests may not provide any information one way or the other about certain faults. Each fault in each outcome of a test also has attached information about how this test outcome affects the probability of this fault.

A test also has associated with it a set of *monitoring actions* and *diagnostic actions*. A monitoring or diagnostic action is an attempt by the agent to gain information from the world. Specifically, a monitoring action represents an increase in the rate at which a given parameter, which is always available to the agent at some frequency anyway, is monitored. A diagnostic action is a request to obtain a single value of a particular parameter from the world. Both types of actions have two types of costs associated with them. There are temporal costs—the amount of time that it takes for the action to be performed and the results to come back. Because of the real-time nature of the deadlines the agent is up against, temporal costs are important to the agent. Temporal costs in turn have two components—the time taken in the world for the parameter values to be obtained, and the computational resources consumed by the agent in analyzing the parameter values to determine a particular outcome for a test. The exact value of the temporal costs will therefore vary depending upon the amount of computational resources the agent has to devote to the diagnosis task. There are also physical costs—the amount of physical resources consumed by the action. In general, these two types of costs must be traded off against one another—we could perform all possible actions with great frequency and save on temporal costs, but at great expense regarding physical costs. Similarly, we could save on physical costs by performing only one monitoring or diagnostic action at a time (thereby performing only those actions that are absolutely necessary) but this would increase the temporal costs.

Faults have associated with them a set of *prior probabilities*. Because we assume that when the reactive plan is invoked, at least one fault is present, the prior probabilities must sum to at least 1.0. In the event that we make *single fault assumption*, which in this paper we do, the sum of the priors must be exactly 1.0.

It follows from the above description of the basics of the problem that the reactive plan will be executing a form of *anytime algorithm*. At various points in time, it will request that tests be performed, and the results of these tests will result in a refinement of the differential diagnosis. However, for any differential diagnosis, it is possible to find a corrective action that is better than all others, on average, for this differential. Ideally, such an action would be one that had substantial value for most of the faults in the differential, and as a result it is desirable that the reactive plan be constructed in such a way that it is likely to find differentials along the way that have good corrective actions

for most of their faults. The algorithm is anytime in the sense that as the differential is refined, the value of the best corrective action available is likely to improve as it can be made more specialized for particular faults. When the deadline is reached, the corrective action associated with the current differential can be recommended and performed.

2.1. Formal statement of the problem

The previous section gave an intuitive statement of the problem we are interested in. In this section we will state the problem more precisely. However, in order to make the problem precise and tractable, we will make first a number of simplifying assumptions which will be adhered to throughout the remainder of the paper.

2.1.1. Simplifying assumptions

(1) *There is only trivial uncertainty handling.* That is, at all times in the diagnosis process, the probabilities of all faults in the differential diagnosis are either zero or in the same proportions that they were in originally.
(2) *Deadlines are hard.* The utility of a given response for a given fault is constant up until a particular hard deadline for that fault, and is zero thereafter. The value of the hard deadline is not, however, necessarily known precisely in advance.
(3) *Temporal costs of tests are constant.* The amount of time taken for a given test to come back is known in advance.
(4) *The agent knows when it has reached its (hard) deadline.* When the hard deadline referred to in number (2) has been reached, the agent will know, by means not discussed in this paper, and can therefore act.
(5) *The single fault assumption is made.* Exactly one fault will be present for a given invocation of the reactive planner.
(6) *Tests provide information only about faults, not about deadlines.* The agent will not learn any additional information about how much time it has left by performing a test beyond what it can infer from the information about faults revealed by that test.

2.1.2. The problem

The reactive planner is given a 9-tuple with which to construct a plan:

$$\langle F, P, D, A, V, T, O, C, S \rangle.$$

These are defined as follows:

- F: the set of faults.
- P: the set of prior probabilities. P is a function mapping F to the interval $[0, 1]$.
- D: the set of deadline distributions. D is a function mapping F to the set of functions, D_f, such that

$$\int_0^\infty D_f(t)\,\mathrm{d}t = 1.$$

Here the semantics are that if $D(f) = D_f$, then the probability that the hard deadline for fault f lies in the interval $[t_1, t_2]$ is:

$$\int_{t_1}^{t_2} D_f(t)\mathrm{d}t.$$

- A: the set of actions.
- U: the matrix of utilities. U is a function mapping $F \times A$ to the interval $[-1, 1]$. The semantics of this will be discussed in more detail below.
- T: the set of tests.
- O: the set of outcomes of tests. O is a function mapping T to sets of faults. For example, if $O(\tau) = \{o_1, o_2, \ldots, o_k\}$ is a set of outcomes, then each o_i is a subset of F corresponding to one possible outcome of the test τ.
- C: the set of temporal costs of tests. This is a function mapping T to the set of nonnegative real numbers.
- S: the set of physical costs of tests. This is a function mapping T to the set of nonnegative real numbers.

It will be noted that the notions of monitoring and diagnostic actions have been collapsed into a single notion of tests. A test consists of a set of monitoring and diagnostic actions, together with the necessary analysis of the results of those actions. The interesting properties (from our point of view) of these monitoring and diagnostic actions can be captured in the temporal costs and the physical costs of tests.

We now describe the goal of the agent vis-a-vis the reactive plan. The agent's goal, roughly speaking, is to recommend a series of tests

$$\langle t_0, \tau_0 \rangle, \langle t_1, \tau_1 \rangle, \langle t_2, \tau_2 \rangle, \ldots$$

where $t_0 \leqslant t_1 \leqslant t_2 \leqslant \cdots$ are the times at which tests $\tau_0, \tau_1, \tau_2, \ldots$ are recommended. All times are measured from the invocation of the reactive plan. When making its recommendation at time t_k, the agent will have available to it the outcomes of all tests that have come back already—that is, all outcomes to tests τ_i, $i < k$, where $t_i + C(\tau_i) \leqslant t_k$. The agent must also recommend (and perform) at the deadline some action a. The agent is assumed to know when the deadline is and to recommend and perform an action at that point, but it continues to perform diagnosis after reaching the deadline unless and until it reaches a complete diagnosis.

The action a is a function of the time t at which the deadline occurs. It also depends on two things—the particular fault that occurs and the outcomes to the various tests (noting that in the case of multiple outcomes including the same fault, all are equally likely). Letting f denote the fault and o denote the various possible outcomes to the tests, we can see the action is actually a function of three variables: $a(t, f, o)$. Thus, the utility is a step function:

$$U_{outcome}(t, f, o) = U(a(t, f, o), f).$$

This function intuitively represents the improving utility of the action obtained to date during a single invocation of the reactive plan. By taking the mean over all outcomes

of the test consistent with this fault, we can reduce this to a function of two variables: $U_{test}(t, f)$ which should also be a step function, albeit with many more steps. Now we can reduce this to a function only of the fault itself by weighting this value function by the deadline distribution function for this fault:

$$U_{fault}(f) = \int_0^\infty D_f(t) U_{test}(t, f) \mathrm{d}t.$$

Finally we may compute the overall performance of the agent by calculating the weighted sum over all faults:

$$U_{tot} = \sum_{f \in F} U_{fault}(f) P(f).$$

This number, U_{tot}, represents the overall performance of the agent on this reactive plan. Two different reactive plans may therefore be compared by computing different values for U_{tot} and comparing. Intuitively, what we have done is: take the various actions recommended by the agent at different points in time and determine the values of those actions for a particular fault. These values are then plotted on a graph and then the average is taken of the different performances that are possible for a single fault because of uncertainty in test outcome. We then determine the average performance over time of the agent on this fault by computing the integral weighted by deadline distribution. Finally, we compute the average performance of the agent over all faults by taking the average of these functions weighted by probabilities of the faults. This is illustrated graphically in Fig. 1. This shows the improving performance of the utility of the action recommended for a fault over time as the action changes from $A0$, $A1$, $A2$, through $A3$. Because in this case the hard deadline is assumed to have a uniform distribution between 2.5 and 3.5, only the utilities of the actions recommended at those time points are relevant, and the integral computing overall performance is calculated over that interval.

So far the analysis has taken into account only the utility of performing actions. What of the cost of performing tests as described in the last section? The temporal cost of performing tests is taken into account when we compute the time axis of these graphs. The physical cost of performing tests will itself be a step function of time, in this case guaranteed to be nondecreasing, which can be computed for a single invocation of the reactive plan as follows:

$$S_{outcome}(t, f, o) = \sum_{t_i \leqslant t} S(t_i).$$

As with values, we can compute the average over all test outcomes consistent with the particular fault to give $S_{test}(t, f)$. Now we can compute the weighted integral over time to give:

$$S_{fault}(f) = \int_0^\infty D_f(t) S_{test}(t, f) \mathrm{d}t.$$

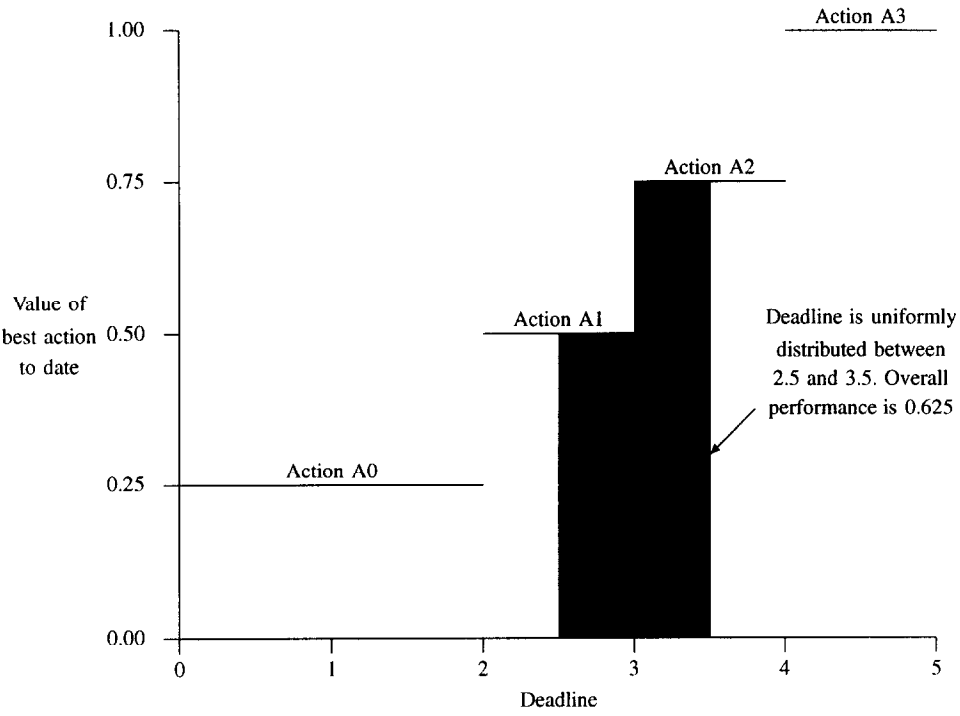

Fig. 1. Reactive plan performance on a single fault.

Finally, we can compute the overall average physical cost of this reactive plan by computing the weighted average over different faults:

$$S_{tot} = \sum_{f \in F} S_{fault}(f) P(f).$$

The true performance of a reactive plan may then be computed by determining the utility-cost difference $U_{tot} - S_{tot}$.

Our goal in this paper is not to directly produce reactive plans. It is, rather, to produce a reactive planner that will generate good reactive plans. Thus, we will have a space of problems $\{\langle F, P, D, A, V, T, O, C, S\rangle\}$. By computing the average performance of the reactive plans produced by different reactive planners for the same problem, we will be able to compare reactive planners. It is our goal in this paper to propose and evaluate a set of reactive planners on just this criterion.

2.2. *Related work*

The most obvious attempted solutions to this type of problem to appear in the literature are the *anytime planning* work of Dean and Boddy [14], the decision-theoretic

analysis of Horvitz [27], and the concept of *bounded rationality* elucidated by Russell and Wefald [37]. Anytime planning involves the idea of an *anytime algorithm* which returns an answer regardless of how long the algorithm runs for, but the value of the answer improves over time. Dean and Boddy's contribution is to analyze the properties of anytime algorithms in general but not to propose a particular anytime algorithm; this paper will follow in this general framework but will present a particular anytime algorithm whose properties can be analyzed. Horvitz and Russell and Wefald do a good job of analyzing the problem from a theoretical point of view. However, the goals of these researchers all differ somewhat from the present research goals in that they assume that deliberation or metareasoning time is nontrivial. In other words, they devote significant computational resources to deciding what to do, rather than actually doing it. In the current research it is an important goal, because of the urgent nature of the domains the agent is expected to operate in, to keep metareasoning time at a trivial level. Any computational resources used by the agent should be devoted exclusively to analyzing the results of tests, not deciding what test to perform next. We are willing to incur a potentially high computational expense at the reactive planning stage in order to save a much smaller amount of time in executing the reactive plan itself. The approaches mentioned above are unable to make this tradeoff because they assume that a single agent does both deliberation and acting, and so time spent deliberating is time lost acting.

Although we characterize this work as a diagnosis problem, the connections between it and the existing diagnosis literature do not seem too close. Classically diagnosis has been cast as a problem in first-order logic [35]. This approach requires that one have a model of the domain in order to do diagnosis. As such Reiter offers one significant advantage over our work—the ability to do first principles diagnosis—and has one significant disadvantage—meeting a real-time deadline is impossible. Reiter also shows that diagnosis is greatly simplified when the *single fault assumption* is made—a conclusion that we agree with.

On the subject of multiple faults, the seminal paper is [16]. De Kleer and Williams take the approach that diagnosing multiple faults is computationally expensive and so they provide a number of techniques for reducing the computational complexity of the search. For example, they suggest beginning the search with small potential conflict sets and then expanding the conflict sets to produce minimal sets that actually explain the flawed behavior of the system. This works when doing diagnosis from first principles but seems to be a hard idea to apply to our problem. Our agent starts with a large differential diagnosis and then gradually works it down to smaller sets of faults. However, we agree with their basic observation that if the number of possible faults is small, then coping with their interactions remains manageable, but we cannot expect the problem of diagnosing all possible subsets of the set of all faults to be computationally tractable.

Other researchers such as Cooper [11] have addressed the problem of computational complexity of diagnosis without looking at meeting real-time deadlines. For example, Cooper has shown that inference in belief networks is NP-hard. The response to this is to attempt to control the search; the TOP N algorithm [25] can be used to generate only the n most likely diagnoses. In addition to the lack of meeting real-time deadlines, this

approach differs from ours in the use of a Bayesian belief net to represent uncertainties. The current work is not an attempt to deal with uncertainty, although it might be a worthwhile possible extension to attempt to do so.

Other approaches to diagnosis include model-based reasoning [13], heuristic classification [8], qualitative reasoning [22,31], decision trees [33], and various techniques in the medical AI literature (e.g. [41]). None of these approaches attempt to deal with the problem of diagnosis in real time, and so they are of limited usefulness to us, although an adaptation of Quinlan's decision trees will prove to be central to our approach as we describe it in the next section.

Another large body of literature of interest to us is the work on *reactive* and *real-time planning*. Before describing the relationship of this work to the present paper, we should define what we mean by these terms as different authors may use them for different purposes. For us, reactive planning is the task of planning *in advance* response to situations that may arise in real time for which there will not be adequate time to respond if we have to do substantial planning at run time. The work in this paper fits our definition of reactive planning.

The ultimate example of reactive planning is *universal planning* [40] where the reactive planner attempts to enumerate in advance exactly what to do in *all* situations the agent may encounter. Ginsburg [23] has shown that in general universal planning is not computationally tractable, although it may be in specific cases. We do not attempt to enumerate all possible situations in advance. Rather, we enumerate a set of faults [12] which we expect to be desirable to respond to (including, for example, those that are likely to be *critical*, but not so critical that response is hopeless), and then plan to respond only to that set of faults. We therefore are attacking a narrower problem than Schoppers did, but with greater chance of success.

A much less extreme example is *triangle tables* [21] where the agent has a plan to execute, but also has a series of preconditions to each step in the plan, so that it knows when the situation has changed unexpectedly so that the plan no longer applies and thereby the agent can replan. This work represented one of the first efforts in reactive planning, but it is limited to recognizing when a plan no longer applies as opposed to providing good actions to perform in such a case.

Brooks' *subsumption architecture* [5] is part of this general body of work, although he would probably not classify his work as reactive planning. The approach involves dividing a robot's task into a number of layers, with the lower layers providing simple abilities such as avoiding objects, and the higher layers providing more complex abilities such as identifying objects. This allows reactivity at the lower levels while still permitting more complex operations at a high level. Perhaps Brooks' most controversial claim is that there is no need for a central control structure in a robot. Although Brooks has provided convincing evidence that his approach works for a simple robot with two or three layers, it is not clear that it can scale up.

Kaelbling and Rosenschein have authored a number of papers on *situated automata theory* [28–30,36]. This theory is based on the assumption that an agent can accurately track and represent the state of the world in real time. The basic idea that they have is that they can provide guaranteed response in a single (constant time) cycle. This seems to imply that their approach would be useful if all deadlines were constant (or at least

multiples of a constant) but in the case of our problem, not only do the deadlines vary in a much richer way, the costs of tests may not correspond exactly to cycle time either, so fitting our problem into their approach seems difficult. Their approach seems far more likely to be successful when agent behavior can be mapped into *levels of competence* such that at each level response is required within a constant cycle time. It simply is not obvious that our problem can be divided up in this way.

Chrisman and Simmons [7] introduce the notion of *sensible planning* which brings decision theory into the analysis of reactive planning. A robot often needs to perform sensing actions in order to diagnose the state of the world. Chrisman and Simmons use decision theory to determine which of several possible *sensing procedures* involves the minimal expected cost. For simple worlds, their approach is probably quite effective, but they have no notion of meeting real-time objectives—they are simply interested in using the best possible sensing procedure.

A number of approaches have been taken to *real-time planning* in the literature. Of these, perhaps Dean et al. [15] attack the problem closest to the work in the present paper. They compute a *policy* (a mapping from states to actions) on the basis of which they can maximize the future discounted value, averaged over time, of the world state. This problem is similar to ours in the sense that states in their view correspond to differential diagnoses in ours, and a policy corresponds to a decision of ours to perform a test in a certain situation (their notion of *action* corresponds to our notion of *test*; our notion of *action* can be used to compute their notion of the *value of a state*). However, our problem differs from theirs in several important respects. Firstly, all possible states in our world correspond to the set of all subsets of our fault set; for a reasonable number of faults this would be intractable. Second, there seems to be an unstated assumption that the time to progress from one state to another is constant in their domains; in our domains it is not. Finally, they seem to require specific mathematical properties of the value discounting function to make their algorithm tractable. This function corresponds roughly to our notion of deadline distribution, which means that their approach only works for one particular type of deadline distribution.

Hendler and Agrawala [24] describe a system in which they unify a *dynamic planning* system with a real-time operating system (MARUTI). Dynamic planning systems are able to both react and plan, and the amount of time devoted to the two tasks itself varies dynamically based upon the nature of the task. Dynamic planning is designed to address the main flaws in both classical planning and reactive planning: it does not assume that a complete plan can be laid out in advance with no unforeseen difficulties, and yet it also does not require anticipation of all possible situations in advance, either. Hendler criticizes existing planning systems on the grounds that they assume that the planner has complete knowledge of the world. He therefore provides a reaction component to handle those situations that the planner could not foresee in advance. However, we go a step further by not assuming that even the reaction component has complete knowledge of the world. But we do not provide any dynamic replanning either, although we anticipate that an agent using our reactive plans will also have other reasoning techniques in its arsenal that will be capable of dynamic replanning.

3. Action-based hierarchies

In this section we will describe the approach, which we call *action-based hierarchies*, that we take to solve the problem outlined in the last section. We will first describe the approach as it applies to a simplified version of the problem, for illustrative purposes, and then extend the approach to deal with the complete problem. We will also discuss the feasibility of the approach along several dimensions. First, the time complexity of the reactive plan must be very low. Second, the space complexity of the reactive plan must at least be manageable. Finally, the time complexity of the reactive planner, although not central to our goals, must be low enough to permit the use of our approach (and possibly to allow some dynamic replanning).

The simplified version of the problem that we will solve first makes the following set of assumptions:

(1) Tests have no physical costs.
(2) No more than one test can be in progress at a time. If the agent performs a test, it must wait for the outcome of that test before performing the next test in its reactive plan. The first simplifying assumption tends to vastly increase the number of tests that can be performed; this assumption limits the number.
(3) Deadline distributions are not taken into account. As such we cannot make absolute claims about the performance of a reactive plan unless it performs better than its competitors for *all* possible values of the deadline. Otherwise we can merely observe which deadline values one reactive plan is better for, and which other deadline values another plan is better for.
(4) The reactive plan is invoked as soon as the fault appears (the previous assumption requires this assumption to also be made).
(5) The fault sets associated with the outcomes of each test are disjoint (that is, for any given fault, there is exactly one possible outcome of each test).

The basic data structure that is used to represent a reactive plan is known as an action-based hierarchy. An example of an action-based hierarchy is shown in Fig. 2. Action-based hierarchies have the following properties:

(1) The basic structure of an action-based hierarchy is similar to that of a decision tree.
(2) The leaf nodes of the hierarchy correspond to individual faults (unlike in a decision tree where they may correspond to classes of faults one wishes to diagnose).
(3) Each higher-level node has associated with it a set of faults that is the *union* of the sets of faults associated with its children.
(4) The top node has associated with it the set of all faults.
(5) Each node has associated with it a corrective action, which ideally will have substantial utility for all faults in the associated fault set, but if no such action exists, may simply have substantial utility for the most likely such faults.

The central idea behind an action-based hierarchy is that by identifying actions with sets of faults, if the agent only has time to make a partial diagnosis, it will be able to perform the action associated with the set of faults related to that partial diagnosis, and derive the benefit associated with that action. The goal, ideally, is still to perform

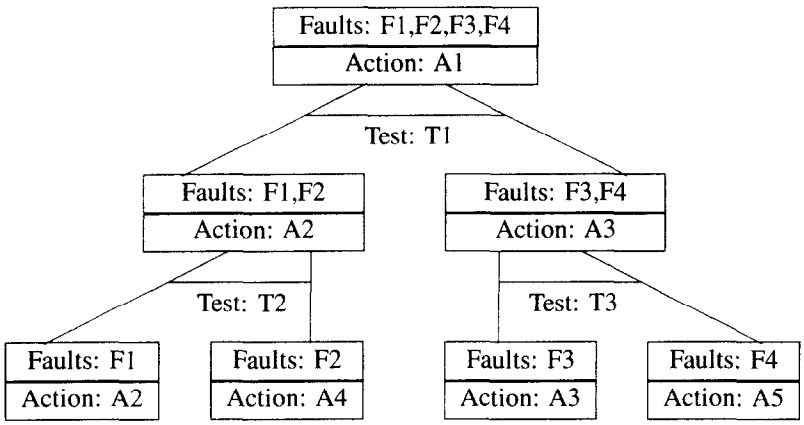

Fig. 2. Sample action-based hierarchy.

complete diagnosis, but it is also necessary to be able to provide good response in a suboptimal situation where complete diagnosis is not possible.

As shown in Fig. 2, there is also associated with each nonterminal node in the hierarchy a test from the set T (defined in the last section). Suppose that node n has associated with it a test τ, and $O(\tau)$ (the set of outcomes of the test τ) $= \{o_1, o_2, \ldots, o_k\}$, and that node n has an associated fault set F_n. Then n will have up to k children in the hierarchy, one corresponding to each outcome o_i for which $o_i \cap F_n$ is nonempty. We can thereby formally define an action-based hierarchy as follows:

(1) The hierarchy itself consists of a pair $\langle R, N \rangle$ where R is the root node of the hierarchy and N is the set of all other nodes.

(2) Each node n in the hierarchy is a 4-tuple $\langle L, \tau, a, F_n \rangle$ defined as follows:
   (1) L is the set of children nodes of this node. In the case of a terminal node, this would be empty.
   (2) τ is the test associated with this node (or the empty set in case L happens to be empty and we are dealing with a terminal node).
   (3) a is the action associated with this node (how this is computed will be explained below).
   (4) F_n is the set of faults associated with this node.

The property given above relating a node to its children may be formally stated as follows: If a node $n = \langle L, \tau, a, F_n \rangle$ has children $L = \{n_1, n_2, \ldots, n_i\}$, with each child n_i having the form $\langle L_i, \tau_i, a_i, F_{n_i} \rangle$, and the test τ has k outcomes $\{o_1, o_2, \ldots, o_k\}$, then for each o_k for which $o_k \cap F_n \neq \emptyset$, there exists exactly one child node n_j for which $F_{n_j} = o_k \cap F_n$.

Having defined the relationship between a node and its children, we now state the conditions that define the action for each node. Specifically, we always associate with a node in the hierarchy the action that gives the best expected utility for the set of faults associated with that node in the hierarchy. In formal terms, we can define the utility of an action for a set of faults by extension from the definition for a single fault:

$$U(F_n, a) = \frac{\sum_{f \in F_n} P(f)U(f, a)}{\sum_{f \in F_n} P(f)}$$

where $P(f)$ denotes the probability of fault f. Next, we can formally define the property that the best action, a_n, at a particular node in the hierarchy must have: if the node is $n = \langle L, \tau, a_n, F_n \rangle$, then

$$U(F_n, a_n) = \max_{a \in A} U(F_n, a).$$

In this case we write

$$a_n = a(F_n).$$

Having defined the best action for a node, and thus the action-based hierarchy, we can now specify the algorithm that constitutes the reactive plan that the agent will use:

Algorithm 1. Reactive plan.

Step 1. Set the *current best hypothesis* to the root node, R, in the hierarchy.
Step 2. Perform the test associated with the current best hypothesis.
Step 3. If the test result does not come back before the deadline is reached, go to Step 5. Otherwise, modify the current best hypothesis to the child of the current best hypothesis corresponding to the outcome of the test that occurs.
Step 4. If the current best hypothesis is a terminal node, recommend the action associated with that node and stop. Otherwise, go to Step 2.
Step 5. Recommend the action associated with the current best hypothesis and stop.

Thus, the reactive plan is completely determined once the agent knows the hierarchy structure. This property of the reactive plan is designed to meet the criterion described in Section 2 that the computational resources consumed in executing it should be minimal. Given this property of the hierarchy structure, the reactive planner needs to be able to structure the hierarchy. It is structured using the following algorithm:

Algorithm 2. Hierarchy structuring algorithm (*The reactive planner*).

Step 1. Start at the top of the yet-to-be-built hierarhcy, associating the set of all faults with this top node.
Step 2. Pick a leaf node in the hierarchy in DFS (depth-first search) order, or stop if there are no more leaf nodes to expand. Associate with this leaf node the action that has the highest expected utility for the set of faults associated with this node.
Step 3. If the leaf node cannot be expanded further, go back to Step 2 and pick another leaf node.
Step 4. Find all tests relevant to the set of faults associated with the current node.
Step 5. If no tests were found in Step 4, go back to Step 2 and pick another leaf node.
Step 6. Determine which test found in Step 4 is best according to some heuristic.
Step 7. Expand the current node with one child corresponding to each possible outcome of the test found in Step 6, and the associated fault sets suitably adjusted.
Step 8. Go back to Step 2 and pick one of the children of the current node.

The structuring described above is very much like that used in structuring a decision tree. The only part of this algorithm that is not completely specified is in Step 6—the heuristic used to determine which test is optimal. There are several possible *hierarchy structuring heuristics* which could be used.

3.1. Hierarchy structuring heuristics

A hierarchy structuring heuristic is a function that maps tests to the real numbers, within the context of a particular node in the hierarchy, such that the larger the result, the more desirable it is to perform at the node. The simplest heuristic that could be used is the information-theoretic heuristic used in structuring decision trees [33]. Specifically, let the set of faults associated with the current node be F_n, and compute the following function of the test τ:

$$\sum_{o \in \tau} P(o) I(o)$$

where

$$P(o) = \sum_{f \in o \cap F_n} P_w(f), \qquad I(o) = \sum_{f \in o \cap F_n} \frac{P_w(f)}{P(o)} \log\left(\frac{P_w(f)}{P(o)}\right)$$

and

$$P_w(f) = \frac{P(f)}{\#\{o \in \tau : f \in o\}}.$$

This heuristic maximizes the information content gained as one moves from a parent to a child node in the hierarchy. Hence it would be expected, in general, to take a minimal number of tests to move from the root node to a terminal node in the hierarchy. However, it does not take into account either the cost of performing tests, or the utility of the best action available at the intermediate nodes in the hierarchy. Thus, we would expect that we could do better—in terms of the evaluation criteria proposed in Section 2—by using a structuring heuristic that does take these factors into account.

Therefore, we propose an *action-based* hierarchy structuring heuristic as follows:

$$\frac{U(\tau, F_n) - U(F_n)}{C(\tau)}$$

where

$$U(F_0) = \max_{a \in A} U(F_0, a)$$

for any set of faults F_0 gives the definition of utility for a set of faults, and

$$U(\tau, F_n) = \frac{\sum_{o \in \tau} U(o \cap F_n) P(o)}{\sum_{o \in \tau} P(o)}.$$

It will be noted that this structuring heuristic takes into account only temporal costs of tests, not physical costs. As such, it relies rather heavily on a particular assumption

made at the beginning of this section—that no more than one test can be in progress at a time. If more than one test were allowed to be in progress at a time, and physical costs of tests were not taken into account, then the agent would be advised to perform as many tests as quickly as possible, as they have no intrinsic cost. In the next section we will relax this assumption at the same time as we take physical costs into account.

This gives two possible heuristics for structuring the hierarchy; there are others which are given in [1]. One can hypothesize about the behavior of an agent executing a reactive plan derived from either of these heuristics. One would expect the agent that is using the action-based heuristic to find actions of higher value for shorter values of the deadline. However, the information-theoretic heuristic would converge faster on terminal nodes, so for higher values of the deadlines this heuristic would be better. Another hypothesis could be that the overall performance (when integrated over all values of the deadline) is likely to be better for the action-based heuristic. As we will see, it will be difficult to prove these hypotheses formally except in a few very specialized cases, but experimentally we can and will test these hypotheses later in the paper.

At this point a word on nomenclature is appropriate. An *action-based hierarchy* is a type of *decision tree*. There are two heuristics being proposed to structure an *action-based hierarchy*: the *action-based heuristic*, and the *information-theoretic heuristic*. Although it is certainly reasonable to hypothesize that the action-based heuristic, being specifically designed for the action-based hierarchy, will provide better hierarchy performance than the information-theoretic heuristic which has a more general applicability to all decision trees, this hypothesis must be verified theoretically and/or experimentally. The hypothesis could fail if the planner using an action-based heuristic makes "greedy" choices at a high level in the hierarchy which do not particularly help it in making the more specific diagnoses at a lower level in the hierarchy.

3.2. *Multiple tests at a time*

We now seek to relax the first and second assumptions stated at the beginning of this section. Specifically, the planner now takes the physical costs of tests into account in deciding when to perform a test. The idea is that the reactive agent may wish to perform a test in advance of its really being needed so as to avoid the delay in acting that would be involved if it had to wait for the test's outcome later on. That is, when the agent reaches a particular node in the hierarchy in its diagnosis process, it may decide to perform not only the test associated with that node but also the tests associated with one or more of its children or other descendants. The process by which it does so is known as *test promotion* because it involves "promoting" a test so that it is performed earlier than it is really needed. The hierarchy is first structured using the deadline distribution heuristic described in [1] (deadline distribution is essential to take into account in deciding whether to do test promotion). Then the test promotion process is done. Test promotion involves deciding whether the cost in performing a test earlier than necessary is greater than the benefit gained by performing it early. The basic algorithm is fairly simple:

Algorithm 3. Test promotion.

Step 1. Structure the hierarchy using the algorithm given above.

Step 2. Set the current node to the root node in the hierarchy.

Step 3. First recursively do test promotion by executing Steps 2–4 with the current node being set to each of the children of the current node.

Step 4. Then for each child of the current node in turn, determine whether the performance profile will improve or be worse if the tests for that child are moved to the current node. If the profile will improve, move the tests up.

A formal description of test promotion may be found in [1].

3.3. Nondisjoint test outcomes

The assumption has been made to date that the outcssumption has been made to date that the outcomes for a test are always disjoint, or alternatively that each test is a partition of the set of faults or that any test has exactly one possible outcome for a given fault. In this section, we examine what happens when this assumption is relaxed to the weaker assumption that a test may have multiple outcomes consistent with a given fault, but those outcomes will all be equally likely in the presence of that fault.

The primary difference that this imposes is that now it is possible for a node in the hierarchy (which corresponds to a set of faults) to have multiple parents, whereas before a node could have only one parent. It will therefore be necessary for the agent, in executing the reactive plan, to keep track of the set of tests that have been executed in reaching a particular node in the hierarchy. One example where this could happen is if we are trying to determine the likely winner of a presidential nomination. One easy test is to determine whether any candidate has a majority of delegates yet. The set of faults here is the set of all candidates, and if this test fails to show a candidate with a majority, then the set of possible faults will remain the set of all candidates. However, the agent needs to know that the "check for majority" has been performed so as not to perform it again—just knowing the set of possible faults is not enough. Furthermore, each node will have associated with it a set of tests, instead of just a single test, and when the node is reached, the highest ranking test on that list that has not previously been executed will be performed. Each node will have a number of sets of children, one set corresponding to each of the tests that can be performed at this node in the hierarchy.

Structuring the hierarchy will proceed similarly as before. The only difference will be that when a child node is potentially to be created, the reactive planner will look at the list of existing nodes and determine whether a node already exists with the same set of faults. If so, no new child node will be created, but rather a pointer will be created to the existing child node. The planner will then determine whether there is a test to be performed at this child node consistent with this trajectory through the hierarchy. If there is not, further expansion of the child node will be necessary; if there is, then no further expansion is necessary. Step 6 in the hierarchy structuring algorithm will change to determine the best test among those that have not already been performed in reaching this node in the hierarchy.

Formally, the only difference is that the 6-tuple that defines each node now becomes:

$$\langle\langle L_1, \ldots, L_k\rangle, \langle\tau_1, \ldots, \tau_k\rangle, a, F_n, \langle E_{n1}, \ldots, E_{nk}\rangle, \langle Q_{n1}, \ldots, Q_{nk}\rangle\rangle$$

so that essentially each node now has a set of children sets instead of just one.

One important assumption has been made in order to reduce the possibility of a combinatorial explosion in expanding the search space. This regards the prior probabilities of lower nodes in the hierarchies. Because of the differing numbers of competing outcomes that different faults may have for a given test, the prior probabilities of faults in lower nodes in the hierarchy may not be in the same proportion as the original prior probabilities of the faults. However, the above approach assumes that they are.

3.4. Related work

The idea of varying the heuristic in structuring decision trees is not new. What is new about this application of decision trees is that they are being used not merely to do classification, but also to provide meaningful responses in the interim when a complete classification is impossible. Since the heuristics used here are tailored to this application, they have not been previously used in the literature. It is because of this substantially different application that we prefer a different term, action-based hierarchies, for our decision trees.

There is a fairly extensive body of literature devoted to exploring different heuristics for decision tree structuring. An information-theoretic heuristic is used by various researchers: by Cheng et al. in GID3 [6], by Fayyad in GID3* [19], by Quinlan in C4 [34], by Breiman et al. in CART [4], and by Clark and Niblett in CN2 [9]. Fayyad and Irani introduce a class separation approach (C-SEP in [20]) where the heuristic measures not the information content of a test but the degree to which it separates faults into the different classes. This presupposes, however, that the goal of using the decision tree is to diagnose down to the level of an individual class, not an individual fault. It is unclear, therefore, how this approach could be applied to the current problem without some delineation of classes that would in turn require applying the action-based heuristic first. GID3 and GID3* differ from ID3 (the original information-theoretic approach of Quinlan) in that only some of the outcomes of a test are used in branching. Many other approaches have also been taken in the literature.

A novel approach to structuring decision trees, *oblique decision trees*, is introduced by Murthy et al. [32]. The idea here is that their algorithm, OC1, determines the optimal oblique hyperplane with which to split the set of data points. The equivalent problem for us would be to design our own tests, rather than using a pre-defined set of tests from a given domain. This would be an interesting topic for further research, but is beyond the scope of the current paper.

4. Formal results

Ideally, one would be able to prove formally that the approach presented in Section 3 represents the best possible solution to the problem given in Section 2. If this could be

done, the paper could immediately be concluded without the need for any experiments or examples! Although it turns out this cannot be done, in this section two formal results will be presented without proof. The first result shows that under certain well-defined assumptions, the action-based hierarchy provides the best performance in the strong sense that no matter what the value of the deadline and no matter what other structuring heuristic might be used, the action-based hierarchy will perform at least as well. The second result gives a similar conclusion in the case where the hierarchy is very small.

4.1. Optimal action-based hierarchies

The assumptions that are required to make the claim that action-based hierarchies provide optimal performance are the following:

(1) All the assumptions made for the most basic version of the problem described in Section 3 must be made.
(2) In addition, one of the following two conditions must hold:
    (1) The temporal costs of all tests must be equal.
    (2) There must be no time wastage at the deadline: that is, the deadline does not occur during the middle of performing a test but rather immediately after a test result comes back.
(3) The really strong assumption that must be made is that the utilities of tests are additive for each fault. The utility of a group of tests for a fault is the amount by which the utility of the best action available for that fault improves after performing the group of tests versus what its utility is before performing the group of tests. Additivity means that the utility of any group of tests is the sum of the utilities of performing each test.

4.2. Very small hierarchies

A similar claim can be made—that action-based hierarchies provide better performance than any possible competitor—under the following conditions which mandate a very small hierarchy:

(1) All the assumptions made for the most basic version of the problem described in Section 3 must be made.
(2) The temporal costs of all tests must be equal.
(3) At most two tests are required to complete the diagnosis process (hence the resulting hierarchy has depth at most two).

5. Abstract experiments

As we saw in Section 4, the set of conditions under which we can formally prove that action-based hierarchies provide the best performance is quite limited. Thus, in this section and the next, we shall seek other means of validating this work. In fact, there will be three separate approaches described in these two sections. In this section,

the values of the inputs to the hierarchy structuring algorithm will be assigned randomly without regard to any specific domain, and the performance derived by using the action-based heuristic versus the information-theoretic heuristic will be compared. If the performance under the action-based heuristic is found to be better by a statistically significant margin than that of the information-theoretic heuristic, then this will be evidence in favor of using this approach for structuring hierarchies in real-time domains.

The validation method described in this section can both be criticized and praised on the grounds that it is independent of any particular domain. To answer that criticism, in the next section we will explore validation of the approach in a particular domain—intensive care unit patient monitoring.

Specifically, in this section we wish to evaluate the following two hypotheses:

(1) Using the action-based hierarchy will provide substantially better performance than the decision tree when evaluated as described in Section 2.

(2) Using the decision tree will provide substantially better performance when only speed in reaching a leaf node matters.

5.1. Experimental design

The problem size, prior probabilities, test outcomes, and values of actions for faults were assigned as follows:

Problem size

The problem size was allowed to vary and was equal to the number of faults, actions, or tests in the domain. That is, the number of faults, actions, and tests were all kept equal, and this number is what was allowed to vary.

Prior probabilities

The difficulty in assigning prior probabilities is that although they must be assigned randomly, they must also sum to 1. The following method was used to assign a set of n prior probabilities (for n faults) $p_1, p_2, \ldots, p_n$: First a set of $n-1$ random variables uniformly distributed on the interval $[0,1]$ was assigned by a random number generator:

$$x_1, x_2, \ldots, x_{n-1}.$$

Then these numbers were permuted so that they were in nondecreasing order:

$$x_{i_1} \leqslant x_{i_2} \leqslant \cdots \leqslant x_{i_{n-1}} \quad \text{where } \{i_1, i_2, \ldots, i_{n-1}\} = \{1, 2, \ldots, n-1\}$$

and finally the priors are assigned as follows:

$$p_1 = x_{i_1}, p_2 = x_{i_2} - x_{i_1}, \ldots, p_{n-1} = x_{i_{n-1}} - x_{i_{n-2}}, p_n = 1 - x_{i_{n-1}}.$$

The important property of this distribution is that the distribution for each prior probability generated in this way is the same.

Tests

The number of tests, n, must be even. For each test, the fault set is randomly partitioned into two disjoint subsets of size $n/2$. This does not appear to offer any bias in favor of either approach. Both the action-based and information-theoretic approaches will tend to favor partitions which give a roughly equal split of the fault set, so in partitioning the fault set in this way, a roughly equal advantage is given to each approach. The partition is assigned randomly from among the $\binom{n}{n/2}$ possible ways of doing this.

5.1.1. Values of actions

The value of an action for a fault is uniformly distributed in the interval $[0, 1]$.

In performing these experiments, we added another hierarchy structuring heuristic in addition to the basic information-theoretic and action-based heuristics: a random heuristic. The random heuristic selects a test at random from the set of all *relevant* tests that can be performed at a given node. Formally, this heuristic is defined as follows:

$$R(t) = \begin{cases} \text{uniform } [0, 1], & \text{if } \exists o \in t: \emptyset \subset o \cap F_n \subset F_n, \\ 0, & \text{otherwise.} \end{cases}$$

5.2. Statistical validation

In order to make any claims about the statistical validity of the results derived from these experiments, we need some means for computing statistical validity. The method used will be the t-statistic [26]. Under reasonable assumptions about the experiments, and with sets of 100 trials, the t-statistic indicates that a difference of 1.75% in hierarchy performance between two heuristics corresponds to one standard deviation and 3.5% corresponds to two standard deviations.

5.3. Experiment results

Fig. 3 shows the difference in performance, over time, of action-based (ABH), information-theoretic (DT), and random heuristics, using a problem size of 64 faults. This shows that the action-based heuristic always does at least as well as the other approaches, no matter what the value of the deadline (the amount of resources available for diagnosis), but that this advantage varies over time from nil at time zero, to a maximum for deadline values of around 3 or 4, back to nil for large deadline values. This corresponds, roughly speaking, to the fact that all hierarchies perform equally well at their root and their leaf nodes, but that at the intermediate nodes there is an advantage for certain hierarchies over others. The advantage appears to be about 8% for deadline values of 3 or 4, which as noted in Section 5.2, corresponds to at least 5 standard deviations, which can be taken to be statistically significant.

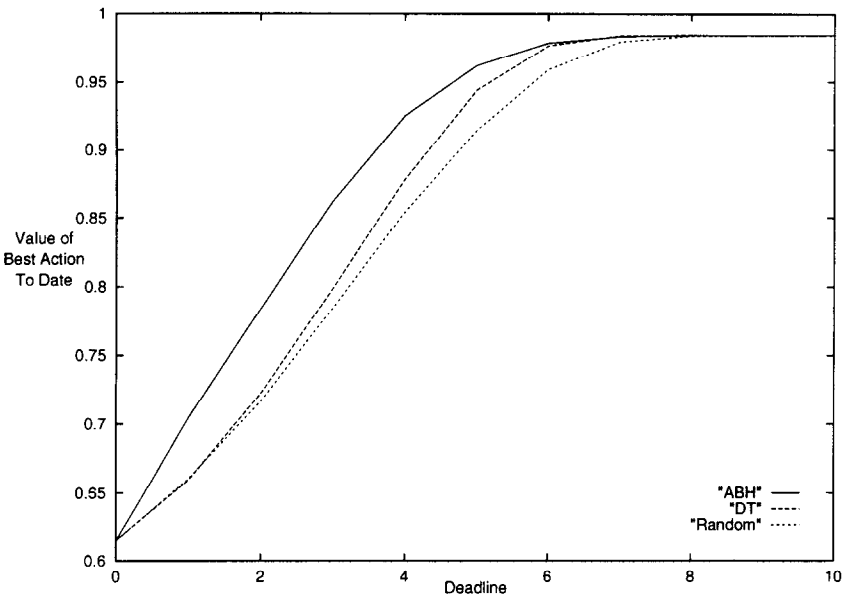

Fig. 3. Hierarchy performance with different deadlines and heuristics.

6. Domain results

The experiments described in Section 5 were successful as far as they went. They provided a good preliminary validation of the approach of using the action-based as opposed to the information-theoretic or random heuristics, as well as the increased benefits that can be realized by taking deadline distributions and test promotion into account. As such, they significantly extended the scope of the theoretical analysis, which was extremely limited in that it required very strict assumptions in order to be provably optimal. However, to date the approach has been completely in the abstract—in this section we apply the approach to a real-world problem to show that similar results can be realized in such a problem.

The first part of this section will describe the domain of application—surgical intensive care unit patient monitoring—and then results from applying the approach to this problem will be presented. These results, while suggestive, cannot be statistically validated because the performance of a single generated hierarchy cannot be statistically significant.

6.1. Intensive care unit patient monitoring

The domain to which these ideas are applied is that of monitoring a patient in an intensive care unit (ICU). A patient in the ICU typically has multiple organ failure and is placed on a ventilator to assist with breathing. Because of this, there can be many problems that might arise that require fast response; this domain is attractive because of

the need to provide such response, and because there are often several different responses available of differing specificity that can be undertaken given differential diagnoses of different sizes.

The areas requiring immediate response can be broadly categorized into three groups: anemic hypoxia problems which generally require a transfusion of some type; oligemic hypoxia problems which may require the immediate invocation of an ALCS (advanced cardiac life support) algorithm or treatment of the underlying cause if more time is available; and oligemic hypoxia problems generally requiring either some tweaking of the ventilator settings or treatment of the underlying cause of the problem. Corrective actions in this domain can be of several types; they can buy a medical practitioner time in order to diagnose the real underlying problem; they can be effective but not necessarily definitive for a wide range of problems; or they can represent the definitive therapy for a particular fault. Because the structuring algorithm requires a particular number for the value of each action, a medical domain expert provided a heuristic estimate of the numerical value of each action for each fault on a scale of 0.0 to 1.0.

The tests available in this domain tended to fit into two large categories—lab tests that require an average of 20–30 minutes to come back—and monitoring actions involving checking parameters—these could be completed in under a minute. The actual tests in this domain require a minimum of computational complexity to be analyzed; this is not a limitation on the action-based hierarchy approach but rather is an artifact of this particular domain.

Many of the inputs for structuring the algorithm will depend on the type of patient in the ICU. For the purposes of illustration, the values used in this demonstration were given by a domain expert based on a typical patient at the Palo Alto Veteran's Administration hospital—a 67 year old male who had just undergone coronary artery bypass graft (CABG) surgery.

Various medical domains have been typical vehicles for applying AI ideas over the years. For example, the KARDIO system [3] precompiles ECG descriptions based on model-based reasoning, which it uses to classify arrhythmias at run time. This represents a solution to half the problem that is solved in this paper; we precompile solutions but also can make tradeoffs at run time in cases where a complete diagnosis is impossible. This appears to be impossible in KARDIO. Another system, VM [18], is similar to the present work in that it is a real-time medical AI system, but it handled real time in a different manner than here. In particular, it was concerned with the complexity of reasoning algorithms and with when data values became stale over time, but not with deadlines. Therefore, it was unable to make the necessary tradeoffs that a system using action-based hierarchies can. Another medical system, TraumAID [10] was able to handle incomplete data, but the notion of meeting a deadline is missing.

Specific to the problem of intensive care unit patient monitoring, there is a fairly wide body of literature. Factor and colleagues [17] developed an architecture known as the *process trellis* and applied it to the problem of ICU patient monitoring. The process trellis consists of a graph of decision processes which execute in parallel; it has the ability to provide parallelism even when knowledge acquisition is done by domain experts with little knowledge of parallelism. Although the ability to do parallel processing would seem to be a major advantage to this approach, the fact that the processes modeled all

directly correspond physiologically to the domain restricts its usefulness in responding to deadlines. Response on deadline requires a component that knows specifically about deadlines and not necessarily about specific organ systems.

An ICU patient monitoring system with a strongly decision-theoretic component is VentPlan [38, 39]. The idea here is to select a decision-theoretic model of physician preferences that effectively trades off complexity and accuracy. This is a different tradeoff than the one made by action-based hierarchies, which trade off time for specificity. Still, it would be interesting to compare the two approaches to see which yields the better results. It appears plausible that action-based hierarchies would perform better in cases where incomplete information is available, while conversely VentPlan would perform better when complete information is available but time is short.

EINTHOVEN [44] is another ICU monitoring system devoted chiefly to the interpretation of ECGs. As such it attacks a different subproblem of the ICU problem than the one discussed in this paper. Other ICU monitoring systems include WEANPRO [42], SIMON [43], and the work of King and colleagues at Vanderbilt [45].

6.2. *Results in the domain*

The hierarchy structuring algorithm was run, with both action-based and information-theoretic heuristics, on the medical problem outlined in Section 6.1. As noted above, it is computationally tractable to generate a complete hierarchy for a set of 58 faults and measure the expected performance of that hierarchy, but the results generated are not statistically significant. Further information about why the generation of a complete hierarchy is computationally tractable can be found in [1]. Of course, the fact that this complete analysis is possible does not mean that there is no uncertainty in the domain, so in that sense we are solving a simplified version of the actual problem with the action-based or information-theoretic heuristics. However, given this simplified approximation to the real problem, a complete analysis of the value of a particular hierarchy is possible.

The basic assumption that is made in moving from the actual problem to the simplified problem is that the various inputs to the hierarchy structuring algorithm are fixed and known in advance. For example, a given test is assumed to come back from the lab in a fixed amount of time; in the actual domain the time would be uncertain and known only to have a certain approximate mean. Similar comments may be made about the other inputs to the hierarchy structuring algorithm.

It should also be remembered that the goal of this research is to compare different possible approaches to the same general problem of reacting when faced with limited resources and only basic associative knowledge about the domain. Other approaches such as model-based reasoning (handling first principles domain knowledge better) or belief networks (handling uncertainty better) might perform better in certain cases especially when the agent has greater resources with which to complete its diagnosis and has the necessary domain knowledge. It is not the goal of the present research to make this comparison, although it would definitely be an interesting topic for further research.

Fig. 4 shows the results for the performance of the medical hierarchy, with value of best action plotted against deadlines. For short values of the deadline, it should be

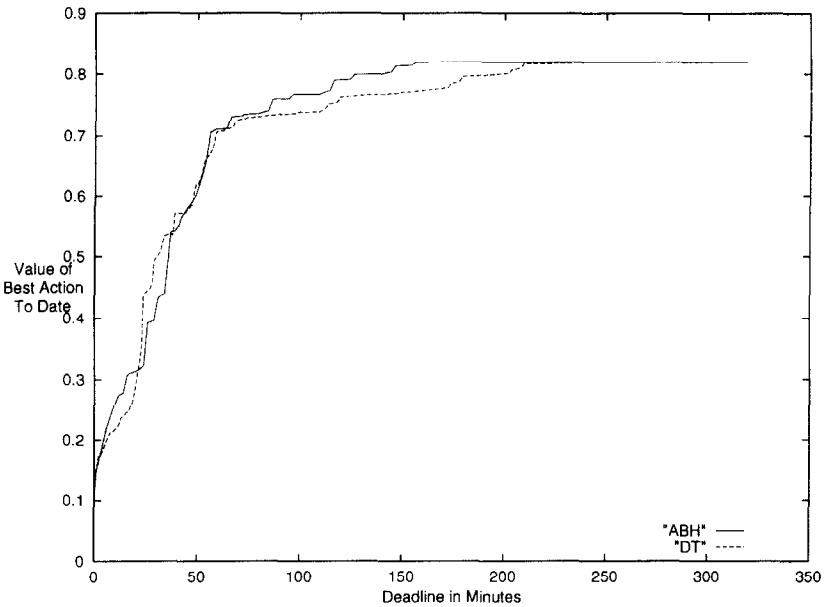

Fig. 4. Hierarchy performance on medical problem (costs factored in).

noted that action-based hierarchies outperformed information-theoretic hierarchies. For some larger values of the deadline, information-theoretic hierarchies do better. These results, which as noted earlier are not statistically significant, seem to suggest that if the deadline is expected to be short, then action-based hierarchies are provided the best response, whereas for larger values of the deadline, the information-theoretic approach may be better.

7. Directions for future research

The work provides an important contribution to the field in several ways. It represents an improvement on existing diagnosis work in that this work either addresses diagnosis but not from the point of view of real-time constraints ([35], [16], and many others), or addresses complexity issues without the specific notion of meeting a real-time deadline ([11] and some others). With regard to existing reactive planning systems, this approach takes us beyond the limitations of having to come up with a response within a single cycle or not at all ([28] and others). Existing anytime systems assume deliberation (deciding what to do next) and action (doing it) are interleaved at run time ([14], [37] and others)—this approach requires that all deliberation take place at planning time. As will be discussed further below, it would be interesting to examine further the limitations and advantages this approach may offer. Existing attribute selection approaches to decision tree construction do not address the real-time question, although it would be interesting to try many of them out to see whether others might do better for the real-time problem than entropy—this issue has not been fully explored.

The question thus arises as to what the limitations are of this approach and whether it is reasonable to believe that further work could push back the limitations. This section will seek to identify a number of the more promising ideas for extension of the work; the description in this section is by no means intended to be complete.

7.1. Reasoning under uncertainty

Perhaps the most important capability that would need to be added to this approach would be the ability to reason under uncertainty. Right now there is only a limited ability in the sense that the system can handle tests that have outcomes that are consistent with more than one fault. However, there is no general ability to handle arbitrary changes in the probability of faults as the reasoning process progresses. The probability of a fault can only be zero or in the same proportion to other possible faults that existed at the beginning of the execution of the reactive plan.

7.2. Deliberation in real time

The assumption underlying much of this research is that it is desirable to precompile solutions in advance to reactive problems, at least as much as possible, because the structuring process is sufficiently complex and computational resources sufficiently scarce in real time that it cannot be wasted on hierarchy structuring. Certainly we are not the only researchers to make this assumption—implicit in the work of Bratko et al. [3] or Widman [44] or any decision tree structuring algorithm this same assumption is made. But is it a reasonable assumption to make? This is something that has not been addressed by this research.

7.3. Decay in values of actions

The value of an action for a particular fault is assumed to be constant up to the hard deadline, and then zero thereafter. This clearly is not valid in general. Indeed, Rutledge [38] has proposed that various models be used for the decay in the value of actions as a function of time. A useful topic of further research, therefore, would be to incorporate Rutledge's (or similar) ideas into this work.

8. Conclusions

This paper has illustrated both the strengths and limitations of the action-based structuring approach. The greatest strength of the approach is that no possible competing approach to the problem previously described in the literature is as effective at solving the problem as the action-based structuring approach. Many of the other technologies described in the literature do not address the problem addressed by this research; their primary drawback is that they are not simultaneously capable of providing response in real time and making the planning/real-time tradeoff so essential to this approach. The one technology that can be applied to this problem—decision trees—proves to be an

effective approach, but capable of being made more so by a number of refinements which have been described in detail in this paper.

In problem domains where reasoning under uncertainty is not necessary, this refined approach represents the best known manner of providing real-time response. This approach could also be combined with known techniques for reasoning under uncertainty in cases where it is necessary, and such a combination would represent an improvement of the state of the art in such cases.

Acknowledgements

This research was supported by ARPA through NASA grant NAG2-581 (ARPA order 8607) and Teknowledge contract 71715 under ARPA contract DAAA21-92-C-0028.

References

[1] D. Ash, Diagnosis using action-based hierarchies for optimal real-time performance, Ph.D. dissertation, Computer Science Department, Stanford University, Stanford, CA (1994).

[2] D. Ash, G. Gold, A. Seiver and B. Hayes-Roth, Guaranteeing real-time response with limited resources, *J. Artif. Intell. Med.* **5** (1992) 49–66.

[3] I. Bratko, I. Mozetič and N. Lavrač, Automatic synthesis and compression of cardiological knowledge, in: J.E. Hayes, D. Michie and J. Richards, eds. *Machine Intelligence* **11** (Oxford University Press, Oxford, 1987) 435–454.

[4] L. Breiman, J. Friedman, R. Olshen and C. Stone, *Classification and Regression Trees* (Wadsworth & Brooks, Monterey, CA, 1984).

[5] R. Brooks, A robust layered control system for a mobile robot, *IEEE J. Robot. Automat.* (March 1986) 14–23.

[6] J. Cheng, U. Fayyad, K. Irani and Z. Qian, Improved decision trees: a generalized version of ID3, in: *Proceedings Fifth International Conference on Machine Learning*, Ann Arbor, MI (1988) 100–108.

[7] L. Chrisman and R. Simmons, Sensible planning: focusing perceptual attention, in: *Proceedings AAAI-91*, Anaheim, CA (1991) 756–761.

[8] W. Clancey, Heuristic classification, *Artif. Intell.* **27** (1985) 289–350.

[9] P. Clark and T. Niblett, The CN2 induction algorithm, *Mach. Learn.* **3** (1989) 261–284.

[10] J. Clarke, M. Niv, B. Webber, K. Fisherkeller, D. Southerland and R. Ryack, TraumAID: a decision aid for managing trauma at various levels of resources, in: *Proceedings Thirteenth Annual Symposium on Computer Applications in Medical Care*, Washington, DC (1989).

[11] G. Cooper, The computational complexity of probabilistic inference on Bayesian belief networks, *Artif. Intell.* **42** (1990) 353–405.

[12] V. Dabija, Deciding whether to plan to react, Ph.D. dissertation, Computer Science Department, Stanford University, Stanford, CA (1994).

[13] R. Davis and W. Hamscher, Model-based reasoning: troubleshooting, in: H. Shrobe, ed., *Exploring Artificial Intelligence: Survey Talks from the National Conference on Artificial Intelligence* (1988) 297–346.

[14] T. Dean and M. Boddy, An analysis of time-dependent planning, in: *Proceedings AAAI-88*, St. Paul, MN (1988) 49–54.

[15] T. Dean, L. Kaelbling, J. Kirman and A. Nicholson, Planning with deadlines in stochastic domains, in: *Proceedings AAAI-93*, Washington, DC (1993) 574–579.

[16] J. de Kleer and B. Williams, Diagnosing multiple faults, *Artif. Intell.* **32** (1987) 97–130.

[17] M. Factor, The process trellis software architecture for parallel real-time monitors, Ph.D. dissertation, Yale University, New Haven, CT (1990).

[18] L. Fagan, VM: representing time-dependent relations in a medical setting, Ph.D. dissertation, Computer Science Department, Stanford University, Stanford, CA (1980).
[19] U. Fayyad, On the induction of decision trees for multiple concept learning, Ph.D. dissertation, EECS Department, University of Michigan, Ann Arbor, MI (1991).
[20] U. Fayyad and K. Irani, The attribute selection problem in decision tree generation, in: *Proceedings AAAI-92*, San Jose, CA (1992) 104–110.
[21] R. Fikes, P. Hart andized robot plans, *Artif. Intell.* **3** (1972) 251–288.
[22] K. Forbus, Qualitative process theory, *Artif. Intell.* **24** (1984) 85–168.
[23] M. Ginsburg, Universal planning: an (almost) universally bad idea, *AI Mag.* **4** (1989) 40–44.
[24] J. Hendler and A. Agrawala, Mission critical planning: AI on the MARUTI real-time operating system, in: K. Sycara, ed., *Proceedings Workshop on Innovative Approaches to Planning, Scheduling, and Control* (1990) 77–84.
[25] M. Henrion, Search-based methods to bound diagnostic probabilities in very large belief nets, in: *Proceedings Seventh International Conference on Uncertainty in AI*, Los Angeles, CA (1991) 142–150.
[26] P. Hoel, *Introduction to Mathematical Statistics* (Wiley, New York, 1947).
[27] E. Horvitz, Reasoning about beliefs and actions under computational resource constraints, in: *Proceedings Third Workshop on Uncertainty in AI*, Seattle, WA (1987).
[28] L. Kaelbling, An architecture for intelligent reactive systems, in: M. Georgeff and A. Lansky, eds., *Reasoning about Actions and Plans* (1987) 395–410.
[29] L. Kaelbling, Goals as parallel program specifications, in: *Proceedings AAAI-88*, St. Paul, MN (1988).
[30] L. Kaelbling, Specifying complex behavior for computer agents, in: *Proceedings Workshop on Innovative Approaches to Planning, Scheduling, and Control* (1990) 433–438.
[31] B. Kuipers, Qualitative simulation, *Artif. Intell.* **29** (1986) 289–338.
[32] S. Murthy, S. Kasif, S. Salzberg and R. Beigel, A randomized induction of oblique decision trees, in: *Proceedings AAAI-93*, Washington, DC (1993) 322–327.
[33] R. Quinlan, Inductive inference as a tool for the construction of high-performance programs, in: R.S. Michalski, T.M. Mitchell and J. Carbonell, eds., *Machine Learning* (Tioga, Palo Alto, CA, 1983).
[34] R. Quinlan, Probabilistic decision trees, in: Y. Kodratoff and R. Michalski, eds., *Machine Learning: An Artificial Intelligence Approach* **3** (Morgan Kaufmann, San Mateo, CA, 1990).
[35] R. Reiter, A theory of diagnosis from first principles, *Artif. Intell.* **32** (1987) 57–95.
[36] S. Rosenschien, Synthesizing information-tracking automata from environment descriptions, in: *Proceedings First International Conference on Principles of Knowledge Representation and Reasoning*, Toronto, Ont. (1989).
[37] S. Russell and E. Wefald, Principles of metareasoning, *Artif. Intell.* **49** (1991) 361–395.
[38] G. Rutledge, Dynamic selection of models under time constraints, in: *Proceedings Second Annual Conference on AI Simulation and Planning in High Autonomy Systems*, Cocoa Beach (1991) 60–67.
[39] G. Rutledge, G. Thomsen, B. Farr, M. Tovar, J. Polaschek, I. Beinlich, L. Sheiner and L. Fagan, The design and implementation of a ventilator-management advisor, *Artif. Intell. Med.* (1993).
[40] M. Schoppers, Universal plans for reactive robots in unpredictable environments, in: *Proceedings IJCAI-87*, Milan (1987).
[41] E. Shortliffe, *MYCIN: Computer-Based Consultations in Medical Therapeutics* (Elsevier, New York, 1976).
[42] D. Tong, Weaning patients from mechanical ventilation: a knowledge-based system approach, *Comput. Meth. Programs Biomed.* **35** (1991) 267–278.
[43] S. Uckun, B. Dawant and D. Lindstrom, Model-based reasoning in intensive-care monitoring: the YAQ approach, *Artif. Intell. Med.* (1993).
[44] L. Widman, A model-based approach to the diagnosis of the cardiac arrhythmias, *Artif. Intell. Med.* **4** (1991) 1–19.
[45] J. Xu, S. Hyman and P. King, Knowledge-based flash evoked potential recognition system, *Artif. Intell. Med.* **4** (1992) 93–109.