Segmenting Reactions to Improve the Behavior of a Planning/Reacting Agent*

Michael Wolverton
Consiglio Nazionale delle Ricerche
Instituto di Elaborazione dell'Informazione
Via Santa Maria, 46
I-56126 Pisa, ITALY
email: michael@iei.pi.cnr.it

Richard Washington
Department of Computer and Information Science
University of Pennsylvania
200 South 33rd Street, 5th floor
Philadelphia, PA 19104-6389, USA
email: rwash@linc.cis.upenn.edu

Abstract

An agent operating in a real-world environment will inevitably encounter some events that demand very quick response—the deadline for an action is short, and the consequences of not acting are high. For these types of events, the agent has a better chance of acting appropriately if it has a pre-stored set of reactions that can be quickly retrieved based on features of the situation. In this paper, we examine the problem of selecting the best set of reactions for an agent to store. In particular, we examine the benefit of including *intervals* of reactions—i.e., executing some reactions only across segments of the complete numeric ranges over which they are defined. We present a decision-theoretic algorithm for selecting the optimal set of reaction intervals, and we present experiments with a computer implementation of that algorithm, called KNEEJERK. The experiments show that the benefit of breaking down reactions into intervals is quite high under a wide range of circumstances.

Introduction

An agent operating in a real-world environment will be faced with many external events that require a quick response. An autonomous robot (Thorpe *et al.* 1991) will have to deal with sudden obstacles posing threats to its own safety and the safety of other entities in its environment. A medical monitoring system (Hayes-Roth *et al.* 1992) will have to deal with sudden changes in the patient's condition necessitating speedy diagnosis and treatment. These types of events often do not allow time for the agent to go through a complicated reasoning process to produce a new or repaired plan given its altered world state. The agent will be much more likely to meet its deadline if it has a pre-computed set of critical events paired with appropriate reactions. In this case, the agent merely needs to retrieve and execute the reaction once it has sensed that the event has occurred. One possible approach to storing reactions would be to store *all* of the system's possible actions as reactions (Schoppers 1987). However, this approach is computationally intractable in terms of both time and space (Ginsberg 1989). One important reason is that the time to retrieve a reaction given a situation will increase with the number of reactions stored, and thus storing more reactions lessens the ability of the agent to react to time-critical events.

Researchers (e.g., (Dabija 1993)) have begun to deal with this difficulty by attacking what we will call the *reaction selection problem*: given a set of events the agent might encounter and a set of actions appropriate for those events, select before execution time a subset of the events/actions for the agent to store as execution-time reactions. This paper reports on our study of two key facets of this problem:

1. How to apply utility theory (recently applied to other planning problems, e.g., (Williamson & Hanks 1994)) to the problem of building a useful reaction set for an agent.
2. Improving the behavior of the agent by dividing reactions into sets of *reaction intervals*, so the agent can take different actions to the same (qualitative) event based on quantitative parameters characterizing that event. This is a crucial ability for real agents for two reasons. First, one reaction often will not be able to effectively handle the entire scope of a critical event. Second, it is frequently difficult or impossible for the programmer of the agent to separate an event into qualitative useful subevents himself.

The sections that follow detail our approach to this problem. After giving an illustrative example of the problem, we present a detailed characterization of the problem and our algorithm for solving it. Then we give the results of experiments run with an implementation of the algorithm, called KNEEJERK. And finally we discuss related work in the literature and possible future directions for this research.

*The first author is supported by an ERCIM Post-Doctoral Fellowship 94-05, and the second author by a Veteran's Affairs Medical Informatics Fellowship, administered by the Philadelphia VAMC. The research reported here was done while the first author was at Rutherford Appleton Laboratory, Chilton, UK, and while the second author was a member of the RFIA group at the CRIN-CNRS & INRIA-Lorraine Laboratory, Nancy, France. Thanks to Vlad Dabija, Cindy Wolverton, and anonymous reviewers for helpful comments on earlier versions of this paper.

Example

Here is a simple example of the reaction selection problem and possible solutions to it. The example, taken from the domain of office robots, considers only a single event and three possible reactions to that event. Both the algorithm and its implementation are designed to handle more complex problems.

In this example, a robot is navigating through a busy hall in an office building while delivering mail. We assume the robot does not have knowledge of the exact locations and trajectories of all the other agents (people and other robots) in the hall, but it might have less precise information such as an estimated number of agents in its area, or an estimated location of the agents immediately in front of it, or an estimated amount of unoccupied space in the area immediately surrounding it. The robot is suddenly faced with an event e: another agent steps out of an office in front of the robot, placing itself in the robot's path. The robot will respond to this event by retrieving a reaction, if it has one appropriate to this event, from its pre-stored cache of reactions. The contents of the cache will have been selected before execution time by a reaction selection module. This module determines the events for which the agent will store reactions, and for each event the particular reaction that will be executed given the specific parameters describing the event.

In this case, the reaction selection module has considered three possible reactions to this general sort of event:

1. r_e^{veer}—the action of making a quick change in direction to avoid the obstacle.
2. r_e^{stop}—the action of stopping as quickly as possible.
3. r_e^{null}—the "action" of taking no reaction to the event and instead relying on run-time replanning to account for the new world situation.

The utility of each of these reactions as a function of time until the collision with the other agent is shown in Figure 1. For very low values of t, the collision is unavoidable and the outcome of any reaction will have low utility. Attempting to stop, and thereby lessening the force of the impact, will always be better than doing nothing when the collision is inevitable, but veering will be more likely than stopping to avoid the collision altogether for very short deadlines (after t_1). However, veering has the additional risk of colliding with other nearby (unseen) agents, and for this reason stopping is preferable for values of t in which it is able to avoid the collision (after t_2). When the collision deadline is very far away (after t_3), doing nothing is the best solution since the agent has time to formulate a plan to get around the new obstacle (or the obstacle has time to remove itself). Performing a sudden reaction like veering or stopping has risks in an environment with moving obstacles, so an unnecessary reaction may be worse than no reaction.

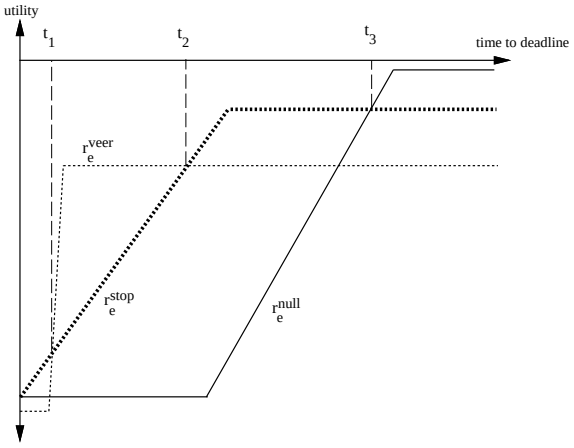

Figure 1: Utility as a function of time to deadline for three possible reactions to a sudden obstacle: Veering out of the way (r_e^{veer}), stopping suddenly (r_e^{stop}), and taking no reaction (replanning—r_e^{null}).

The reactions of Figure 1 are for this example all defined as functions of a single variable, time. However, in principle there may be other continuous parameters describing the event that are available to the agent and that may be used to calculate the utility of reactions. Examples include the estimated amount of free space in the area immediately surrounding it (from which a program can derive the likelihood of causing a collision by veering or stopping) and the agent's own speed (from which it can calculate the expected extent of the damage to the obstacle agent and to itself).

Given an infinite amount of cost-free storage, a reaction selection module would like to include four separate reaction intervals for this event: stopping for values of t from 0 to t_1, veering from t_1 to t_2, stopping again from t_2 to t_3, and the null reaction from t_3 on. In reality, though, storing more reactions at plan time means taking longer to retrieve them at execution time, and that extra retrieval time affects the utility functions of the reactions. Distinguishing among even a small set of reactions may take a non-trivial amount of time, for example if the agent must sense the world to identify the situation correctly. An extra reaction interval added to the stored set of reactions will delay all of the agent's non-null reactions, in effect shifting the graphs for r_e^{veer}, r_e^{stop}, and r_e^{null} in Figure 1 to the right and decreasing the overall expected utility for the reaction set. The cost of any delay in reaction can be quite high in this situation. For this reason, it may not be worth including the interval of r_e^{stop} from 0 to t_1 since the utility gained by adding that interval will not offset the utility lost from delayed reaction.

A general reaction decision module must deal with this problem for many different events and their associated reactions. That is, given a set of events and possible reactions to those events, it must select the

set of reaction intervals to store such that the agent's behavior is optimized. The next section presents a detailed description of this problem.

Model of the Problem

For this problem, we assume an agent that is operating according to a plan in an environment where events can occur suddenly and unexpectedly. The agent's plan is constructed off-line, before execution time. The agent has two methods of dealing with an unexpected event: it can retrieve from its pre-stored cache of reactions the specific reaction appropriate for this event, or, if it does not have an appropriate reaction in its cache (i.e., retrieves only the null reaction r_e^{null} for this event), it can use its replanning capability to dynamically construct a new plan or modify its old plan. The cache is generated before execution time by selecting the *subset* of events to which the agent will react from a given set of possible events, and for each selected event choosing a method of reacting to it. The reaction selection problem, then, is the problem of constructing this cache.

The specific version of the reaction selection problem we are solving has three inputs:

1. A set of *events* E, where each $e \in E$ has associated with it:
   a. The probability of e occurring during some given fixed-length time interval of the agent's execution, $Pr(e)$.
   b. A list of associated *reactions* R_e, which must include the null reaction r_e^{null}. r_e^{null} represents the "action" of not reacting to e but instead relying on the agent's run-time planning to dynamically develop a new plan to account for the new event. Reactions will be described below.
   c. A list of discrete features that uniquely identify the event, S_e (derivable by the agent from its sensor input).
   d. A list of continuous numeric parameters that further describe the event $X_e = x_1, x_2, \ldots, x_n$.
2. A set of possible outcomes O and an associated utility of each $o \in O$. Possible values for utilities will come from a utility space U.
3. A model M of storage/retrieval resource consumption of the agent. This model consists of knowledge of the expected retrieval time given a particular set size, and knowledge of the amount of memory for storing reactions available to the agent. This model is described further below.

A reaction r_e is described by a list of probability densities, each of which gives the probability of a single outcome as a function of e's numeric parameters, given that the reaction has been executed. From this list of probability densities, the probability of e occurring $Pr(e)$, and the given utilities of the outcomes, it is a straightforward application of classical probability theory to produce a function that returns the total utility of r_e's execution given the values of the event's numeric parameters X_e. This function, $f_r : X_e \rightarrow U$, will be referred to as the *utility function* of the reaction r.

The storage/retrieval model is the knowledge of how reaction utility functions change as the size of the store from which reactions are drawn changes. For example, a reaction utility function of a single parameter, time, will tend to be "shifted right" as the storage size (and thus retrieval time) increases. For the functions of Figure 1, as the retrieval time increases, the utility of r_e^{veer}, r_e^{stop}, and r_e^{null} will remain flat at their starting values for a longer period of time, so the overall expected utility of these reactions will decrease. The storage/retrieval model M can then be seen as a function that maps a utility function and a cache size into a new utility function.

For a reaction to an event with n numeric parameters, a *reaction interval* is defined as an n-dimensional region of parameter space over which the reaction is defined, bounded in each dimension by a convex interval (Ladkin 1986) (a continuous interval without holes). More precisely, for a reaction r_e and parameters $x_1, \ldots, x_n$, if r_e is defined over the region described by $x_i \in [x_i^s, x_i^e]$, then that region is a reaction interval of r_e. We will refer to the *total utility* of a reaction interval as the integral of its utility function over the n-dimensional interval that characterizes the reaction interval. That is, for a reaction interval of the reaction r over the interval I, its total utility is given by

$$\int_{p \in I} f_r(p)$$

To summarize, this model of the reaction selection problem requires as input information about possible events that may occur, possible reactions to those events, the expected utility of executing those reactions as a function of continuous parameters (e.g., time), and a model of how storing more reaction intervals affects the reaction's expected utility. The objective of the problem is to take these inputs and output a set of reaction intervals such that the total expected utility of the reaction interval set is maximized. The next section describes an algorithm that solves this problem.

Algorithm

The algorithm for finding the optimal set of reaction intervals is given in Figure 2. It starts with the smallest possible reaction set and steps through the possible set sizes, finding the best reaction set at that size and comparing it to the best found so far. It terminates when the utility of the best reaction set it has found is certain to be better than the utility of all reaction sets larger than the current size. This algorithm is guaranteed to find the optimal reaction interval set provided that (1) the termination condition is satisfied, (2) the function UpperBoundOnUtility returns an upper bound for all

```
function FindBestSet(E, O, M)
   ReactionUtilityFns := CalculateUtilityFns(E, O);
   BestFound := φ;   /* best set of intervals found so far */
   CurSetSize := smallest possible legal set size;
   ReactionUtilityFns := M(CurSetSize, ReactionUtilityFns);

   /* Continue until BestFound is known to be best possible */
   while UpperBoundOnUtility(ReactionUtilityFns, CurSetSize) >
     TotalUtility(BestFound) do begin

       /* Get the optimal set of size exactly CurSetSize */
       New := BestSetOfFixedSize(ReactionUtilityFns, CurSetSize);
       if TotalUtility(New) > TotalUtility(BestFound) then
         BestFound := New;
       CurSetSize := smallest untested legal set size;
       ReactionUtilityFns := M(CurSetSize, ReactionUtilityFns);
   end;
   return(BestFound);
end.
```

Figure 2: Algorithm for finding the optimum set of reaction intervals given (1) E, a set of events and associated reactions, (2) O, a set of possible outcomes and associated utilities, and (3) M, the storage/retrieval model, a function that adjusts reaction utility functions for retrieval time delay based on the size of the stored reaction set

reaction sets, and (3) BestSetOfFixedSize returns the optimal reaction interval set for a given size. We discuss here how these goals can be achieved.

UpperBoundOnUtility(R, n) returns an upper bound on the total utility of any reaction interval set from a set of reactions R whose size is $\geq n$. Our implementation, KNEEJERK, uses a straightforward method of computing this upper bound. In brief, the program computes the bound thusly: For each possible interval find the reaction with the maximum utility over that interval; sum these maximum utilities over all the possible intervals. Assuming that utilities (and likewise UpperBoundOnUtility) are monotonically decreasing as the set size rises (as was the case in the example presented earlier), and assuming that there *is* an optimal set of size $\geq$ CurSetSize (i.e., that the utility of reaction sets does not keep rising forever as the set size grows), this method will in fact return the desired upper bound.These assumptions hold in all the situations we have investigated, and we believe they will hold in nearly all real-world planning/execution situations. The algorithm will terminate if UpperBoundOnUtility eventually drops below the utility of the best set found so far. Again, this condition should hold in all realistic domains, because each increase in the set size will only delay the execution of the reactions, which will diminish the total utility of each reaction, which in turn will lower the value of the UpperBoundOnUtility.

BestSetOfFixedSize(R, n) finds the best reaction interval set from R of size $\leq n$. This is a combinatorial optimization problem for which a number of possible techniques can be applied. Currently, KNEEJERK uses a branch-and-bound search to find the optimal set. We have also implemented a polynomial-time solution using three-dimensional dynamic programming, which allows slightly larger problems to be solved. For very large problems, it may be desirable to use an approximation technique such as simulated annealing (Aarts & Korst 1989).

Most of the other key operations in the algorithm—the method for calculating a reaction's utility function given its event's probability and the probabilities and utilities of the outcomes associated with it (CalculateUtilityFns), the method for calculating the total utility of a reaction interval set (TotalUtility), and the storage retrieval model (M)—were described in the previous section. The method of getting the next legal set size will depend on the storage/retrieval mechanism used. If the reactions will be stored in binary decision tree, for example, the legal set sizes for the nth iteration of the algorithm will be the possible number of leaves in a decision tree of depth n. That is, the algorithm will consider any set size $\leq 2^n$ during the nth iteration.

Experimental Results

This section presents the results of experiments with our implementation of the previous section's algorithm, KNEEJERK. KNEEJERK solves the reaction selection problem for reaction utility functions of one variable, each of which is a continuous, piecewise linear function (a function comprised of connected line segments). So the reaction utility functions shown in Figure 1 are examples of the type that KNEEJERK takes as input.

Our objective in running these experiments was to compare the approach using reaction intervals proposed here with the approach of only allowing a single reaction to an event. We will call these the "With-Intervals" and "Without-Intervals" approaches. Since the Without-Intervals approach is a subset of the With-Intervals approach (the complete reaction is an interval in itself), With-Intervals is always guaranteed to find a solution that is as good or better than Without-Intervals. What we want to learn from these experiments is the level of difference between the two. That is, we wanted to know how much better, on average, an agent would perform given the optimal reaction interval set instead of the optimal complete reaction set.

For the particular experiment reported here, each problem given to the two approaches consisted of 6 events, each of which had associated with it a null reaction plus a number of non-null reactions. The number of non-null reactions to these events was our independent variable in this experiment, varying from 0 to 30 reactions. Each reaction was assigned randomly to one of the six events, and each had associated with it a random utility function. These random utility functions each consisted of 4 random points joined by lines. For each reaction size (0 to 30), 16 different problems were run and their results were averaged. Each problem was

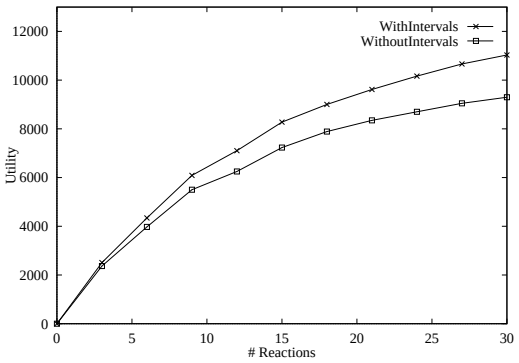

Figure 3: Benefit, compared to not reacting to any event, of two approaches to selecting reactions: Including reaction intervals and including only entire reactions.

run on both the With-Intervals and Without-Intervals approaches, and the total expected utility of the reaction interval set (or reaction set, in Without-Intervals' case) was tabulated. The storage/retrieval model used was that of a complete binary decision tree: the possible set sizes considered were of size (up to) 2^n for all non-negative integers n, and the retrieval for each of those set sizes was modeled as $\mathcal{O}(n)$.

The results of our experiment are shown in Figure 3. The graph shows the expected benefit of each approach—the approach's reaction set's utility compared with the utility of performing the null reaction for each event—as the number of reactions grows. This graph demonstrates two important results. First, it shows that constructing reaction sets with intervals significantly improves the utility of an agent's behavior over constructing sets with only a single reaction covering an entire event. The With-Intervals approach outperformed the Without-Intervals approach by around 20% at the maximum number of reactions we tested (30). Second, it shows that the advantage of With-Intervals over WithoutIntervals grows as the problems become more complex. The percentage difference between the two methods grew steadily as the ratio of reactions to events grew. These two results suggest that the WithIntervals approach is likely to be a useful technique for constructing reaction caches for real-world problems.

We have repeated these experiments for a wide range of event set sizes and reaction set sizes, and in each case we have observed the same general trends reported above. We have also investigated the behavior of the program as the retrieval time changes. That is, if the retrieval time for a reaction interval set of size 2^n is modeled as kn, we have looked at the utilities of reaction sets produced by the two methods as k varies. Here we observed very little change in the differences between the two methods as the retrieval time changed: for the entire range of k-values we investigated, the magnitude of the utility difference between WithIntervals and WithoutIntervals remained relatively stable.

Related Work

The work presented in this paper is built on the work of Dabija (Dabija 1993). With the specific problem we address, we extend that work in several important ways. In our work, the utility value of a reaction is defined as a continuous utility function of parameters characterizing the situation, whereas for Dabija, each reaction has a single utility value. While it is true that each of our reaction intervals could be treated by Dabija as a separate contingency/reaction pair, Dabija provides no mechanism for segmenting the world into these fine-grained contingencies, and presumably requires the agent's programmer to do this. We provide an algorithm that performs this non-trivial segmentation process as a by-product of selecting the optimum set of segments. Besides taking the burden of segmentation off the programmer, this allows the description of the events to the system to be more realistic and natural in at least two additional ways. First, the definition of one event is independent of the other events in the system. Second, each event can have multiple reactions, so our algorithm can solve not only the problem of *what* to react to, but also *how* to react to it in each part of its parameter space.

There are two other important ways in which we have extended Dabija's work. First, we provide an algorithm that finds the set of reaction intervals that optimize the expected utility of the agent's behavior. Dabija provides a greedy algorithm that is not optimal. Second, we use the decision-theoretic utility model for evaluating alternative reaction sets, as compared to a more parameter-specific evaluation model.

Universal plans (Schoppers 1987) organize reactions into a decision tree for retrieval, but there is just one reaction per situation, and all reactions are always included. Also, the cost of retrieval time is not modeled or reasoned about. Ash (Ash & Hayes-Roth 1993) organizes a set of reactions, with potentially multiple reactions per event, into a decision tree so as to optimize their retrieval, but again all reactions are included. We are interested in choosing a subset of reactions such that the overall utility is optimized.

Markov model techniques (Cassandra, Kaelbling, & Littman 1994; Dean *et al.* 1993; Chrisman & Simmons 1991) are designed to find the best solution given the possible states of the world. We are concentrating on how best to divide the world into possible states (where for us this state is the situation associated with each reaction). Another important difference is that we incorporate the resources consumed by adding extra actions (in the form of delayed response time) into our model, thus making utility and resource requirements mutually dependent.

Resource-bounded decision-theoretic planning rea-

sons about comparative utilities in deciding how to expand a plan. In (Horvitz 1988), a model is presented of the utility of partial results under resource bounds; this is used in (Horvitz, Cooper, & Heckerman 1989) to decide between planning and action. A model of partial results is used in (Dean & Boddy 1988) to guarantee incremental improvements over time when planning may be cut off. Utilities can also be used to provide search control for the planning process itself (Williamson & Hanks 1994). In contrast, we use utilities to select a partial set of reactions that will be optimal with respect to the possible events in each situation.

The reaction selection problem can be seen as an instance of the *utility problem* (Minton 1990). One major way in which our work deviates from previous work on the utility problem is that we are considering a computational situation in which the marginal value of including a reaction (or rule) in the agent's store strongly depends on the size and the contents of the store itself. In our situation the decision of what to include in the reaction set must be made globally—considering the reaction set in its entirety—rather than by locally considering heuristic estimates of the utility of each reaction separately.

Future Directions and Conclusion

Our work on reaction selection could be used to guide an agent's perception as well. In real-world situations, sensing is a limited resource, and the agent must choose how much effort to spend on the various sensors (Washington & Hayes-Roth 1989; Chrisman & Simmons 1991). The features that distinguish the particular reactions the agent has stored should be the first ones that the perception module tries to discern. In particular, the boundaries between reaction intervals mark the points where the agent should concentrate its attention so that it can distinguish among situations requiring different reactions. In addition, there may be some parameters that influence the difference in the utilities of reactions more than others. In this case, these parameters should receive more attention so that they are more accurately sensed.

Our work on selecting reactions is designed to be used with an agent that can both plan and react. In general, the reaction model we have presented can be used to define possible actions and outcomes from a given state: the best reaction set is chosen, and then the outcomes are integrated into the plan as possible future states, from which further reactions may be planned. In this way the agent can explicitly reason about the results of its reactive actions.

Also, the agent can learn the probabilities of events and outcomes and the utilities of outcomes from experience. All of the parameters of our approach can be statistically approximated directly from a set of training examples. This approach could also be used to refine an imperfect utility model over time.

In summary, we have presented a formulation of the reaction decision problem and an argument for its importance in real-world agent behavior. We devised an algorithm based on a decision-theoretic utility model of world events and outcomes that is guaranteed to find the optimal set of reaction intervals under realistic circumstances. And we have shown the value of this approach through experiments with an implemented reaction decision module. The experiments show the approach consistently significantly outperforming the approach of including entire reactions.

References

Aarts, E., and Korst, J. 1989. *Simulated Annealing and Boltzmann Machines.* John Wiley & Sons.

Ash, D., and Hayes-Roth, B. 1993. A comparison of action-based hierarchies and decision trees for real-time performance. In *AAAI-93.*

Cassandra, A.; Kaelbling, L.; and Littman, M. 1994. Acting optimally in partially observable stochastic domains. Technical Report CS-94-20, Brown University Department of Computer Science.

Chrisman, L., and Simmons, R. 1991. Sensible planning: Focusing perceptual attention. In *AAAI-91.*

Dabija, V. 1993. *Deciding Whether to Plan to React.* Ph.D. Dissertation, Stanford University.

Dean, T., and Boddy, M. 1988. An analysis of time-dependent planning. In *AAAI-88.*

Dean, T.; Kaelbling, L.; Kirman, J.; and Nicholson, A. 1993. Planning under time constraints in stochastic domains. In *AAAI-93.*

Ginsberg, M. 1989. Universal planning: An (almost) universally bad idea. *AI Magazine* 10:40–44.

Hayes-Roth, B.; Washington, R.; Ash, D.; Hewett, R.; Collinot, A.; Vina, A.; and Seiver, A. 1992. Guardian: A prototype intelligent agent for intensive-care monitoring. *The Journal of AI in Medicine* 4:165–185.

Horvitz, E.; Cooper, G.; and Heckerman, D. 1989. Reflection and action under scarce resources: Theoretical principles and empirical study. In *AAAI-89.*

Horvitz, E. 1988. Reasoning under varying and uncertain resource constraints. In *AAAI-88.*

Ladkin, P. 1986. Primitives and units for time specification. In *AAAI-86.*

Minton, S. 1990. Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence* 42:363–391.

Schoppers, M. 1987. Universal plans for reactive robots in unpredictable environments. In *IJCAI-87.*

Thorpe, C.; Hebert, M.; Kanade, T.; and Shafer, S. 1991. Toward autonomous driving: The CMU navlab. *IEEE Expert* 6:31–52.

Washington, R., and Hayes-Roth, B. 1989. Input data management for real-time AI systems. In *IJCAI-89.*

Williamson, M., and Hanks, S. 1994. Optimal planning with a goal-directed utility model. In *AIPS-94.*