A Survey of Research in Deliberative Real-Time Artificial Intelligence[1]

Alan Garvey and Victor Lesser
Department of Computer Science
University of Massachusetts

UMass Computer Science Technical Report 93–84
November 19, 1993

Abstract

This paper surveys recent research in deliberative real-time artificial intelligence (AI). Major areas of study have been *anytime algorithms*, *approximate processing*, and large system architectures. We describe several systems in each of these areas, focusing both on progress within the field, and the costs, benefits and interactions among different problem and algorithm complexity limitations used in the surveyed work.

[1]This is an invited paper that will appear in the journal *Real-Time Systems* in 1994. This material is based upon work supported by the National Science Foundation under Grant No. IRI-9208920, NSF contract CDA 8922572, DARPA under ONR contract N00014-92-J-1698 and ONR contract N00014-92-J-1450. The content of the information does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

1 Introduction

Historically, the field of AI has not been particularly concerned with real-time performance. The problems tackled by AI systems are generally very difficult and the focus has been on showing how particular difficult problems can be solved using complex algorithms that often have search as a major component. The dependence on search can often lead to unpredictable performance, because of the difficulty of predicting how much of the search space will need to be examined to find a solution. One example of this difficulty has been in the area of planning. Progressively more difficult real world planning situations have been dealt with by progressively more sophisticated planners with correspondingly larger search spaces and more complex search operators. However, with the increasing complexity it became less clear that the performance of such planners would allow them to be used in actual applications.

A realization of this problem, along with a pressing desire to embed AI systems in larger applications (such as robots) led to the development of *reactive* AI systems [Agre and Chapman, 1987, Brooks, 1986, Firby, 1987]. The goal of the reactive AI systems work (at least initially) was to investigate how close you could come to an intelligent agent with just hardwired reactions to the actual external environment. Obviously such agents are able to react very quickly to changes in the environment. However, some balance between quick responses to environmental change and time for deliberation to allow reasoning about what actions to perform seems to offer the best hope for truly adaptive, intelligent agents. One approach is to have two or more very different, asynchronous subsystems (e.g., one highly reactive and one highly deliberative) and combine their responses, for example work on Phoenix [Howe *et al.*, 1990], Guardian [Hayes-Roth, 1990] and CIRCA [Musliner *et al.*, 1993]. An alternative is to take a more integrated approach where there is a single architecture and a range of responses are possible given the time criticality of the required response. Approaches in this area include *flexible computation* [Horvitz and Rutledge, 1991], *deliberation scheduling* [Boddy and Dean, 1993], compilation of *anytime algorithms* [Zilberstein, 1993] and *design-to-time* real-time scheduling [Garvey and Lesser, 1993]. Both of these approaches are known as *deliberative,* because they deliberate to determine how to act (rather than just reacting).

The second alternative is based on the use of approximate algorithms for solving problems. At least two broadly different kinds of approximation algorithms have been used in real-time AI research. They are:

- ***Anytime algorithms***—an iterative refinement approach where an imprecise answer is generated quickly and refined through some number of iterations.
- ***Multiple and approximate methods***—where a number of different algorithms are available for a task, each of which is capable of generating solutions having different characteristics. These algorithms may be more or less appropriate for different characteristics of the problem, and may make tradeoffs of solution quality versus time.

A strict definition of real-time is that the system guarantees that tasks will receive up to their worst-case runtime and resource requirements, which presumably means that tasks will produce the best possible results (highest quality results). In real-time AI the focus is usually on high-level goal achievement, rather than worst-case requirements. For this reason, often in real-time AI less strict definitions of real-time are used. One common (and usually implicit) definition is that the system will statistically (e.g., on average) achieve the required quality value by the required time, but no guarantee is made about any particular task. Additionally, the anytime algorithm and approximate processing approaches have the characteristic that the system nearly always produces *some* quality value for tasks, although the value achieved may or may not be useful, given the criteria for high-level goal achievement.

Because of the difficulties in predicting the expected performance of algorithms for solving AI problems, the (often intractable) worst-case performance of those algorithms, and the cost of scheduling AI task structures, some limitations in complexity are required in real-time AI problem solving. These complexity limitations take the form of reducing the set of problems that can be represented and/or using solution methods that are designed to produce *satisficing* solutions when there is not sufficient time to produce optimal solutions. One way of explaining the research described in this survey is that it is a study of various kinds of complexity limitations: what can be potentially gained by overcoming the limitation, what the cost of overcoming the limitation is, what tradeoffs can be made among various limitations, and conversely what is gained by making particular limiting assumptions. For example, some real-time AI approaches assume that tasks are independent, i.e, how one task is solved has no effect on how other tasks can be solved. This complexity-reducing assumption often allows much more efficient (sometimes optimal) decision procedures for deciding how to allocate time to each task to maximize overall system performance, but reduces the general applicability of those procedures.

Our goal here is to describe some of the more specific complexity-limiting techniques that real-time AI researchers have found to be practically useful. Many of them seem to take the form of *a priori* limitations in the kind of problems that can be represented[1]. This contrasts with a recent paper on real-time AI by Strosnider and Paul [Strosnider and Paul, 1993] that categorizes these systems based on the approaches they use to make search more predictable. The approaches they examine are pruning (reducing the search space), ordering (considering elements in a heuristic order), approximating (not considering all search states, possibly by combining groups of them together), and scoping (focusing search in particular time- and/or space-constrained parts of the search space).

Several areas of progress have been identified in real-time AI research including extensions to the kinds of problems that can be solved in real-time, a focus on the issues of embedding real-time AI components in larger applications, and a movement toward dynamic real-time performance at runtime. Initial work in real-time AI examined techniques for achieving real-time performance in particular components of AI systems, while ignoring (or at least deemphasizing) other components. More recently, the emphasis has been on making all (or at least most) components of a system real-time. As we describe many existing real-time AI systems, we will attempt to balance an understanding of the complexity limitations used by systems with descriptions of progress in the field.

This paper first discusses work that uses the anytime algorithm approach, then describes work involving the use of multiple approximate methods. Next, work is examined that combines these and other ideas together to form larger systems. Finally a summary describes several complexity limitations used by these real-time AI systems, what the potential advantages of those using these limitations are, and how they can be overcome if necessary.

2 Anytime algorithms

An *anytime algorithm* is an iterative refinement algorithm that can be interrupted and asked to provide an answer at any time. It is expected that the quality of the answer will increase as the anytime algorithm is given increasing runtime (up to some maximum quality). The term "anytime algorithm" was coined by Dean and Boddy in the late '80s [Dean and Boddy, 1988], but iterative refinement algorithms with anytime characteristics have been studied in many areas including numerical approximation, heuristic search, dynamic programming, *Monte Carlo* algorithms, and database query handling. Associated with each anytime algorithm is a *performance profile*, which is a function that maps from the time given to an

[1]It should be noted that most of these complexity limitations are also apparent in systems-oriented real-time work, often accompanied by even stronger simplifications (e.g., all computations must have predictable, polynomial-time worst-case performance).

```
ANYTIME-TSP(V, iter)
1       Tour ← INITIAL-TOUR(V)
2       cost ← COST(Tour)
3       REGISTER-RESULT(Tour)
4       for i ← 1 to iter
5           e1 ← RANDOM-EDGE(Tour)
6           e2 ← RANDOM-EDGE(Tour)
7           δ ← COST(Tour) − COST(SWITCH(Tour, e1, e2))
8           if δ > 0 then
9               Tour ← SWITCH(Tour, e1, e2)
10              cost ← cost − δ
11              REGISTER-RESULT(Tour)
12      SIGNAL(TERMINATION)
13      HALT
```

Figure 1: An anytime traveling salesman algorithm

anytime algorithm (and in some cases input quality) to the value (quality) produced by that algorithm.

Figures 1, 2 and 3 and Table 1 provide a detailed example from Zilberstein [Zilberstein, 1993] of what an anytime algorithm is and how performance profiles can be constructed. Figure 1 is a simple anytime algorithm for solving the Traveling Salesman Problem (TSP). This algorithm quickly constructs an initial tour, registers that result (making it available should the algorithm be halted), then repeatedly chooses two random edges and evaluates whether switching them results in a better tour, and if it does updates the registered solution to that new tour. Figure 2 shows the runtime and quality results obtained by running this algorithm on randomly generated inputs and stopping it at a randomly generated time. Figure 3 shows the expected performance profile generated directly from the map data (the weighted means of the quality at discrete times). Table 1 shows the performance distribution profile for this algorithm. Here different values in a row indicate discrete probabilities that such a quality will be achieved in such a runtime. These are the actual values used in most of Zilberstein's compilation algorithms. Somewhat related to the idea of a performance profile is earlier systems-oriented work on scheduling non-anytime tasks where each task has a value function associated with finishing it by a particular time [Jensen *et al.*, 1985]. The goal in this work was to build a schedule that optimized value by finishing tasks at appropriate times.

Anytime algorithms have the advantage of always having an answer at hand, so they can respond quickly to changing environmental situations. They also provide maximum flexibility to control mechanisms, by allowing any incremental amount of extra work to result in a incrementally improved result. These features make anytime algorithms particularly useful in many real-time situations.

Another potential advantage of anytime algorithms is that there is usually one anytime algorithm for each type of task the system has to perform, whereas in multiple method work there may be several methods for each task. Thus only one method has to be encoded, potentially reducing the programming time and reducing the likelihood of errors.

Closely related to work on anytime algorithms is research within the systems community on *imprecise computation*. Imprecise computation is an approach to real-time problem solving that assumes that tasks have a *mandatory* part and an *optional* part. The mandatory part is an uninterruptible computation that must be executed completely to ensure correct performance of the system. The optional part is

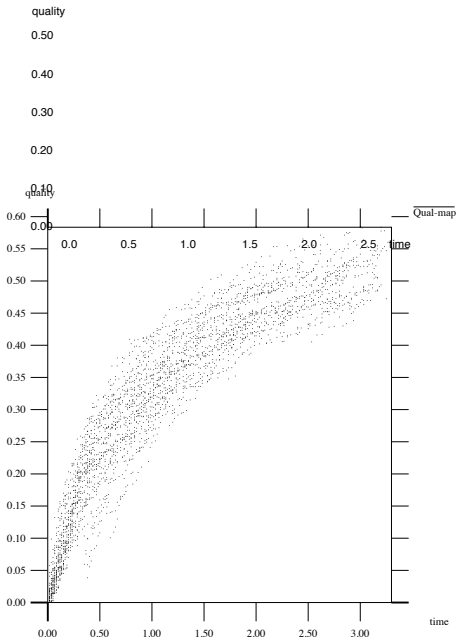

Figure 2: The mapping of quality versus runtime for many runs of a TSP algorithm.

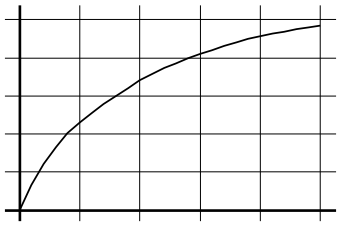

Figure 3: The expected performance profile of the TSP algorithm.

Table 1: The performance distribution profile of the TSP algorithm

	quality											
time	.025	.075	.125	.175	.225	.275	.325	.375	.425	.475	.525	.575
0.0	1.00											
0.2	0.02	0.30	0.48	0.16	0.04							
0.4		0.04	0.12	0.24	0.36	0.24						
0.6			0.04	0.10	0.30	0.34	0.22					
0.8				0.02	0.16	0.34	0.30	0.14	0.04			
1.0					0.02	0.18	0.38	0.26	0.16			
1.2						0.06	0.24	0.40	0.28	0.02		
1.4							0.10	0.40	0.42	0.08		
1.6							0.04	0.30	0.44	0.20	0.02	
1.8								0.10	0.54	0.32	0.04	
2.0									0.44	0.48	0.08	
2.2									0.28	0.52	0.18	0.02
2.4									0.16	0.50	0.30	0.04

(presumably) an iterative refinement algorithm that improves the quality of the result generated by the mandatory part. There is a very large body of work related to scheduling algorithms within this model with various assumptions about characteristics of the tasks to be scheduled [Chung *et al.*, 1990, Marlin *et al.*, 1990, Kenny and Lin, 1991, Shih *et al.*, 1991, Liu *et al.*, 1991a, Liu *et al.*, 1991b, Leung *et al.*, 1992, Ho *et al.*, 1992, Dey *et al.*, 1993].

The emphasis in most of this work has been on finding optimal polynomial-time scheduling algorithms for strongly circumscribed problems within this area. Examples of problems for which such reasonable-performing algorithms have been found include:

- Minimize total error (defined to be the total amount of optional part time not executed) when all task weights are equal and tasks have no mandatory parts.
- Minimize the number of unexecuted optional parts under the 0/1 constraint[2] in the special case where all optional tasks have equal processing time.
- Minimize the number of unexecuted optional parts (again under the 0/1 constraint) when all tasks have the same ready time.

One issue of major importance in imprecise computation research is finding simplifying assumptions that still allow interesting, complex problems to be represented.

Within this section we first present Dean and Boddy's initial work on anytime algorithms [Dean and Boddy, 1988, Boddy and Dean, 1989, Boddy, 1991, Dean and Wellman, 1991, Boddy and Dean, 1993], including their deliberation scheduling algorithms, then we describe Russell and Zilberstein's work on compiling anytime algorithms [Russell and Zilberstein, 1991, Zilberstein and Russell, 1992a, Zilberstein and Russell, 1992b, Zilberstein, 1993]. Next we discuss several projects that use anytime algorithms to solve particular problems, including Horvitz's work on using anytime algorithms in various parts of a decision-theoretic problem-solver [Horvitz, 1988, Horvitz, 1989, Horvitz *et al.*, 1989, Horvitz and Rutledge, 1991], Korf's approach to real-time A* search [Korf, 1990], Collinot and Hayes-Roth's satisficing blackboard control loop [Collinot and Hayes-Roth, 1990, Hayes-Roth and Collinot, 1991], and Ash and Hayes-Roth's anytime diagnosis system [Ash and Hayes-Roth, 1993].

2.1 Dean and Boddy's work

The focus of Dean and Boddy's work in this area has been on what they call *deliberation scheduling* applied to *time-dependent planning problems.* Deliberation scheduling is the explicit allocation of resources to tasks (in most cases anytime algorithms) so as to maximize the total value of an agent's computation. Time-dependent planning problems are defined to be planning problems where the time available for responding to events varies from situation to situation.

In Dean and Boddy's terminology a time-dependent planning problem consists of a set of event types, a set of action types, one anytime algorithm decision procedure for each event type to determine the appropriate action to take, and a value function. More time given to the anytime decision procedure that chooses which action to take will result in a more highly valued action being chosen (up to some maximum value). An agent has knowledge of future events, including the types of those events and the time at which they are expected to occur. The agent constructs a schedule that allocates some amount of time to some of the decision procedures, not allocating any time to decision procedures after the "deadline" of the occurrence of the associated event. The goal is to maximize the sum of the values of the responses to the events. The null response is assumed to have zero value (as opposed to a penalty for missing a deadline). Values of responses are assumed to be independent of one another.

[2] A schedule satisfies the 0/1 constraint if every optional task is either executed to completion or not executed at all.

Given these assumptions, Dean and Boddy describe a pair of algorithms for finding optimal schedules [Boddy, 1991]. (A slightly updated version of these algorithms can be found in a later paper [Boddy and Dean, 1993]). These algorithms primarily differ in their assumptions about the performance profiles. Note that similar, but slightly more general algorithms have been developed independently by Dey et al. [Dey *et al.*, 1993].

The first algorithm (known as *DS-1*), assumes only that the gain for a performance profile at any particular time is available. This algorithm schedules backwards from the latest deadline to the current time, in fixed increments of size δ. It simply assigns each increment to the executable task with the largest gain. After each δ has been assigned, the scheduler goes back and gives each task a contiguous block of time. They show that for any $\epsilon > 0$, there is a $\delta > 0$ such that *DS-1* constructs a schedule within ϵ of optimal.

The second algorithm (known as *DS-2*), assumes that the performance profiles can be described in terms of certain known functions. This algorithm also schedules backward from the last deadline to the current time; this time an increment is the time back to the previous deadline. Within each increment, time is allocated to each task according to the solution to a set of linear equations. The exact structure of the equations depends on the form of the performance profile functions. Examples are given for performance profiles of the form:

$$
\begin{aligned}
f(x) &= \alpha(1 - e^{-\lambda x}),\ \lambda > 0 \\
f(x) &= \begin{cases} \lambda \log x, & x < x^* \\ \lambda \log x^*, & x \geq x^* \end{cases} \\
f(x) &= \text{any general piecewise linear function}
\end{aligned}
$$

In this work, at least in part because of the complexity-limiting assumption of independence of tasks, Dean and Boddy are able to find scheduling algorithms that are computationally inexpensive and have predictable performance. Thus, they do not factor the cost of scheduling into their analysis (although any actual application would have to allow time for scheduling to occur).

2.2 Compilation of anytime algorithms

The focus of Russell and Zilberstein work [Russell and Zilberstein, 1991, Zilberstein, 1993] is based on structuring a computation as the combination of anytime algorithms, where the results of a set of anytime algorithms are inputs to another anytime algorithm. In this framework, the performance profiles for tasks are *conditional* on the quality of their inputs, allowing dependencies between tasks to be explicitly represented. Their goal is given a specific deadline to decide how to allocate time among the individual anytime algorithms so that the expected value of the computation is optimized. They call such compiled programs *contract anytime algorithms*. More generally, a contract anytime algorithm is an algorithm that returns increased value as it is given more time, but must be told in advance how much time it is going to get and may not return any answer if it is given less time than prescribed.

They also extend the concept of performance profile by defining three types of profiles that better represent the uncertainty inherent in such information. Their three types of performance profiles are:

- The ***performance distribution profile*** (PDP) of an algorithm is a function that maps computation time to a probability distribution of the quality of the results. Often this is represented as a table that maps time allocations to expected qualities with discrete probabilities. Table 1 is an example of a performance distribution profile.

- The ***expected performance profile*** (PEP) of an algorithm is a function that maps computation time to the expected quality of the results.

- The ***performance interval profile*** (PIP) of an algorithm is a function that maps computation time to the upper and lower bounds of the quality of the results.

Zilberstein [Russell and Zilberstein, 1991, Zilberstein, 1993] describes several algorithms for compiling anytime algorithms. These algorithms all take a set of dependent (contract or interruptible) anytime algorithms (i.e., a program composed of anytime algorithms) and from them construct a contract anytime algorithm that optimizes system performance for a given amount of available time. These algorithms differ in their assumptions about the form of the dependencies described by the conditional performance profiles. When the dependencies are unrestricted (i.e., the dependencies form a directed acyclic graph) the problem is NP-Complete in a strong sense, and in order to solve such problems he describes three heuristic algorithms that perform adequately in the problems that he has investigated. In the cases of a linear composition of anytime algorithms or tree-structured inputs (i.e., the output of an anytime algorithm is used as input by exactly one other anytime algorithm) he shows that *local compilation* produces globally optimal results. Local compilation involves optimizing the quality of an anytime algorithm by considering only the performance profiles of the anytime algorithms that produce its inputs. Local compilation is analogous to the KNAPSACK problem, which has known pseudo-polynomial time solutions [Zilberstein, 1993]. Pseudo-polynomial time solutions are available for local compilation under the assumption that the tree has bounded degree, (i.e., the number of inputs to each anytime algorithm is bounded). The solutions produced are optimal if each of the conditional performance profiles is a monotonic non-decreasing function of input quality, (i.e., the quality of an anytime algorithm never decreases as its input quality increases).

They also prove that an interruptible algorithm can be constructed from any contract algorithm with the caveat that the quality that the contract algorithm can achieve in time t might take up to $4t$ in the interruptible algorithm. Contract anytime algorithms are useful, because constructing them using compilation is much easier than constructing interruptible anytime algorithms. The result of the compilation process is a contract anytime algorithm that gives specified amounts of runtime to each of its component (contract or interruptible) anytime algorithms. Because the amounts of runtime are directly controlled, the constructed anytime algorithm will never run past its deadline, but, because of uncertainty and unpredictability, it may fail to achieve the quality level required. With different available runtimes passed to the compiler the result could be considered to be multiple approximate methods for meeting a high level goal. For example, a design-to-time scheduler (see Section 3.2) could choose among different compiled contract algorithms at runtime, depending on the current situation.

2.3 Practical applications of anytime algorithms

2.3.1 Horvitz's flexible decision theoretic reasoning

Horvitz [Horvitz, 1988, Horvitz, 1989, Horvitz *et al.*, 1989, Horvitz and Breese, 1990, Horvitz and Rutledge, 1991] has been primarily interested in using anytime algorithms (he refers to them as *flexible computations*) in all aspects of decision-theoretic problem-solving. In decision theory, probabilistic information about the possible outcomes of actions together with knowledge about the utility of those outcomes are used to make the "best" decision possible, where best means the decision most likely to have the highest utility value. Horvitz's work has three main facets:

- determining how to assign utility values to the incomplete partial results returned by an anytime algorithm

- finding anytime algorithms to implement decision-theoretic reasoning about what action to perform next

- finding an optimum balance between time spent doing metareasoning and time spent solving actual problems

As a simple example of how Horvitz assigns utilities to partial results, he considers the problem of sorting [Horvitz, 1988]. Normally, sorting is not considered to be a real-time problem, but consider a sorting algorithm that is part of a larger algorithm that assigns priorities to items in a schedule and wants to order them by priority. In situations where not enough time is available to do the complete sort, it might be useful to use a sorting algorithm that, for example, results in a list that has the first elements of the list (corresponding to the first elements of the schedule) correctly sorted and the later elements less accurately sorted. Horvitz defines a set of dimensions that can be useful in characterizing the value of a partial sort, such as *disorder* (the average distance between current locations and expected final locations for elements), *high/low-end completion* (the contiguous length of positions that contain elements that are in their correct positions), and *bounded disorder* (an upper bound on the distance between current and final position for any element in the list). Using these measures of utility, he shows that Shellsort is excellent at refining bounded disorder and selection sort excels at refining low-end completion. When the particular utility measure used involves low-end completion (as in the real-time situation mentioned above), it is often the case that selection sort has a higher expected utility than, say mergesort, for a large range of expected resource availabilities, despite the worst-case performance of selection sort being $O(N^2)$ versus $O(N \log N)$ for mergesort. That is, the utility of using selection sort can be higher than the utility of using mergesort, if sorting must sometimes stop before completion.

Based on such an approach to assigning utilities to partial and incomplete computations, Horvitz has been developing anytime algorithms for doing decision-theoretic metareasoning. He describes desirable properties that such inference techniques would possess including:

- *Flexibility*—meaning the ability to react gracefully to a broad range of resource availabilities and problems. Aspects of flexibility include *value continuity* (the value of the decision returned by a decision procedure should continuously range from 0 to 1 depending on the resource allocation to that procedure), *value monotonicity* (the value of the decision returned by a decision procedure should be a monotonically increasing function of the amount of resource the procedure is given), *convergence* (the value of the decision returned by the procedure should converge on the optimal complete value given enough resources), and *value dominance* (procedures should have ranges of time over which the value of the decision returned increases monotonically as the amount of resource increases—termed *value-dominant intervals*).

- *Bounded Optimality*—meaning the optimization of utility given assumptions about expected problems and limitations on available resources. In particular he expects a reasoning system to have a repertoire of strategies to choose from, and defines *bounded strategic optimality* to be the application of strategies from this repertoire so as to maximize expected utility, given probability distributions of the costs and benefits of applying various strategies.

Traditional decision theory has the desirable property that it finds provably optimal solutions to problems. Unfortunately the updating of beliefs using probabilistic inference is NP-Hard [Cooper, 1990], making traditional decision theory unsuitable for most real-time situations. Horvitz describes several approaches for approximating the probabilistic inference component of decision-theoretic problem solving. Many of these approximations have the characteristic of providing better answers as they are given additional runtime (i.e., they are flexible computations). These include:

- *bound calculation and propagation*—finding upper and lower bounds on probabilities rather than traditional point values.

- *stochastic simulation*—techniques for characterizing probability distributions by a process of weighted random sampling.

- *completeness modulation*—reasoning about what aspects of the entire problem model to include in reasoning. For example, dependencies in a belief network can be assigned importance values that capture the usefulness of including these dependencies in belief calculations. Only those dependencies above some dynamically-calculated importance threshold are included in calculations.

- *abstraction modulation*—use different problem models that represent the problem at different levels of abstraction. Presumably it is more efficient to reason at higher levels of abstraction.

- *local reformulation*—modify specific troublesome topologies in a belief network. This is probably most useful at design-time.

- *default reasoning and compilation*—having canned solutions that may be indexed by specific problem attributes. These are expected to be useful mainly in extremely time-constrained situations.

One example of this work in practice is described by Horvitz and Rutledge [Horvitz and Rutledge, 1991]. An approach known as *bounded conditioning* is used to approximate the propagation of beliefs through a belief network. They describe experiments using this technique to compute probabilities in a belief network from a medical application. They show that their system can balance the expected value of solving additional simple network problems with the cost of delaying taking action; the exact behavior in a particular situation depends on the relative utilities of the outcomes.

Horvitz has also looked at the general problem of partitioning resources between metareasoning and base-level problem solving [Horvitz and Breese, 1990]. In particular he examines how much time to devote to solution planning for utility-directed problems where both planning and base-level computation is carried out using anytime algorithms. He first considers the case of what he calls *ideal reflection* where you can optimally calculate the amount of time to compute before acting, given an anytime algorithm that increases the utility value over time and a function that describes the cost of delay. He shows how this problem can be solved using simple mathematics when the performance profile of the anytime algorithm and the cost of delay functions have particular forms. He then extends his model to represent a metareasoning process that can modify the performance profile of the anytime algorithm (i.e., reduce the time that it takes the anytime algorithm to improve the utility value to a particular level). He finds solutions to this problem as well, making assumptions about the functional descriptions. Finally he looks at the special case of minimizing the amount of time required to reach a predefined utility value. Again he finds mathematical solutions, assuming particular forms for the various functions. In all of these cases what might be called the metametareasoning costs (the costs of making the calculations about how much time to spend metareasoning versus base-level reasoning) are a small constant, because they simply involve plugging particular values into a simple mathematical function, where which mathematical function to use depends on the forms of the various functions. In general he is able to show that calculations concerning the partitioning of resources between metareasoning and base-level problem solving are quite simple in at least the situations he describes. (Of course, obtaining functions that have the characteristics he assumes may be much more difficult.)[3]

[3]Russell and Wefald [Russell and Wefald, 1991] investigate similar problems (partitioning resources between metareasoning and base-level problem solving) without assuming the availability of anytime algorithms in either the metareasoning or base-level computations. They avoid the metameta- problem by making what they call a *meta-greedy* assumption of just choosing the next single step with the highest utility (rather than considering

2.3.2 Real-time A* search

RTA* [Korf, 1990] is a real-time search algorithm that effectively solves normal state-space search problems using a contract anytime approach. The basic idea is to interleave moving down what appears to be the best path so far with refining the idea of what the best path is. The refining of the best path is done using a simple search algorithm that searches to a fixed depth. The fixed depth is chosen depending on the amount of time allowed for each move, which is determined using a heuristic estimate of the number of moves required to get to a goal state and the total amount of time allowed. The result is that the total search time and the search time per move are tightly controlled, but the quality of the result depends crucially on the accuracy of the heuristics used to estimate distance from a goal state and the ability of limited depth search to recommend moves in the direction of a goal state.

2.3.3 Satisficing anytime blackboard control

Guardian [Hayes-Roth, 1990] is a medical monitoring application built in a blackboard architecture (BB*), and, as such, has an agenda-based control mechanism. However, the standard BB* control cycle is not well-suited to real-time performance. It chooses the best action to perform by elaborating all possible actions, rating each of those against all rating heuristics, then choosing the one with the highest rating. This can take a long time, and will vary in duration depending on the number of actions being considered and the complexity of the control heuristics. Collinot and Hayes-Roth [Collinot and Hayes-Roth, 1990, Hayes-Roth and Collinot, 1991] have devised a satisficing anytime control cycle to replace the standard cycle. In the satisficing cycle, actions most likely to be rated highly are processed first, and as soon as an action is found that is good enough or the time limit for control reasoning has run out the best action found so far is recommended. They experimented with this cycle in the Guardian system and showed that it significantly improved the runtime performance of the control cycle, with only occasional bad outcomes relating to choosing a satisficing rather than optimally chosen action.

2.3.4 Anytime diagnosis

One of the high level tasks of the Guardian system is diagnosing and recommending treatment for medical conditions in the patients being monitored. To perform this task Guardian has an anytime diagnosis component [Ash *et al.*, 1993, Ash and Hayes-Roth, 1993]. This work defines *action-based hierarchies*, which are similar in structure to decision trees. Each node in a hierarchy is a collection of faults, with an associated action to take. Links to subnodes are based on tests that discriminate among faults. Processing starts at the root of the tree with all known possible faults and some default action, and progressively moves down the tree refining the set of faults by performing tests. At anytime this procedure can be interrupted and will return the action associated with the most specific diagnosis currently available. If enough processing time is allowed, the system should zero in on the particular set of actual current faults and the best recommended action for those faults.

3 Multiple Methods

In the multiple method approach, instead of having a single anytime algorithm that produces better results as it is given additional time, a set of methods is used that makes tradeoffs in duration versus quality and may have different performance characteristics in different environmental situations. This approach was originally elucidated in the late '80s by Lesser [Lesser *et al.*, 1988] and termed *approximate processing*.

sequences of computations). They argue that the resulting decision-theoretic metalevel exhibits anytime algorithm behavior.

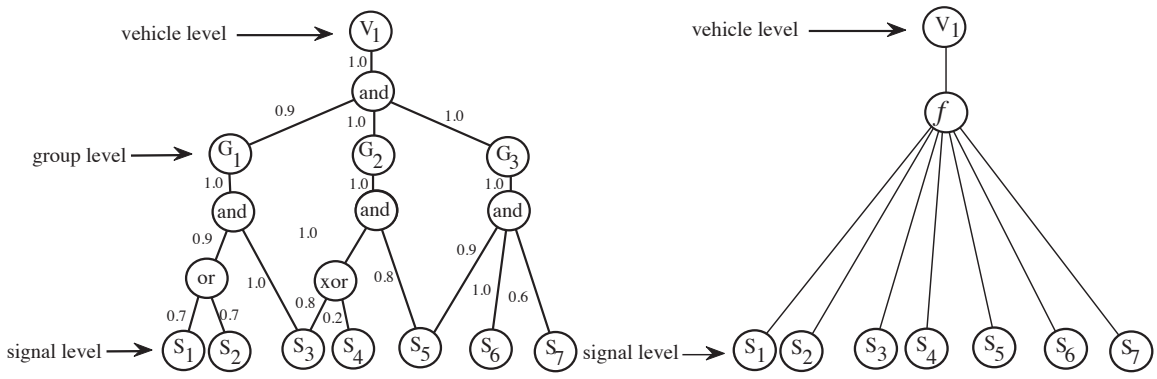

Figure 4: Both complete (on the left) and approximating (on the right) grammars describing the hypotheses necessary to identify a vehicle of type V_1 in a sensor interpretation application.

Figure 4 provides an example from Decker et al. [Decker *et al.*, 1993] of multiple methods for a sensor interpretation application. This figure shows two grammars describing characteristics of hypotheses necessary to identify a particular vehicle type. In the case on the left a number of intermediate objects are hypothesized from the low level sensor data, then combined together to hypothesize a vehicle object. In the case on the right, the intermediate level is bypassed and a vehicle is hypothesized based just on the low level sensor data. This second method is able to hypothesize the vehicle more quickly, but with reduced certainty and precision.

The multiple method approach has at least two potential advantages over an anytime algorithm approach. One is that it does not rely on the existence of iterative refinement algorithms that produce incrementally improving solutions as they are given increasing amounts of runtime. Clearly such algorithms exist for some classes of problems, but just as certainly there are problems that will be difficult if not impossible to solve in an anytime fashion. The difficulty is in finding algorithms whose expected quality improves in a predictable, monotonic fashion.

Another potential advantage of the multiple method approach is that multiple methods do not just make quality/duration tradeoffs, they can be entirely different approaches to solving the problem. These approaches can have very different characteristics depending on particular environmental situations. That is, the quality/duration tradeoffs made by multiple methods can be very different in different environmental situations. Alternatively, in most anytime algorithm work, the assumption is made that one anytime algorithm is available that is expected to work effectively in all environmental situations.

The multiple method approach has been examined in at least two contexts in the systems-oriented real-time community. One is in imprecise computation work with the 0/1 constraint (where the optional part has to be executed to completion or not at all). This is equivalent to having two methods, one that executes just the mandatory part and one that executes both the mandatory and optional parts. The other place where multiple methods appear in systems-oriented real-time work is in error handlers, which have been discussed as a way to recover when tasks cannot be scheduled. In this case there are also two methods, one that performs a task normally and one that does some (presumably very fast) recovery when a task cannot be executed normally.

Less work has been done on the multiple method approach, probably because of its less obviously appealing theoretical properties (i.e., its discrete rather than linear performance characteristics.) This section first describes work by Lesser *et al.* on approximate processing [Lesser *et al.*, 1988, Decker *et al.*, 1990, Decker *et al.*, 1993], then moves on to later work by Lesser and Garvey on *design-to-time* real-time scheduling [Garvey *et al.*, 1993, Garvey and Lesser, 1993]. Next work is described by Bonissone and Halverson on solving dynamic classification problems [Bonissone and Halverson, 1990] and Etzioni [Etzioni, 1991] on the usefulness of marginal utility in scheduling. Finally work on the ABE/RT

real-time system building tool is described [Lark *et al.*, 1990].

3.1 Approximate processing

Approximate processing is an approach to real-time problem solving in situations where satisficing answers are acceptable and some combination of data, knowledge, and control approximations are available. For approximate processing to be successful it necessary to have useful approximations with predictable effects for the application of interest, a problem solving architecture that allows these approximations to be represented reasonably, and control mechanisms for making decisions about which approximations to use. One important aspect of approximate processing is having a representation that allows approximations to be interoperable (i.e., allows approximations to be mixed and matched, rather than each approximation having its own representation that is incompatible with the others).

Initial work on approximate processing [Lesser *et al.*, 1988] focused on generating approximations in a disciplined way in an existing application. This work describes several approximations for use in doing complex signal interpretation tasks. These approximations are categorized as *approximate search strategies*, *data approximations* and *knowledge approximations*. They examine the usefulness of these approximations in the Distributed Vehicle Monitoring Testbed (DVMT).

Decker et al. [Decker *et al.*, 1990] describe extensions to a standard blackboard architecture to support approximate processing. These extensions include ways to represent data approximations by clustering data points together, parameterizing of knowledge sources to allow them to be used for approximate and complete processing, representations for belief and uncertainty of data, and a new, more explicitly controllable low-level control loop containing *filters* to allow data to be filtered from one step to another, *mappings* to control how hypotheses map to goals and goals map to knowledge sources, and *mergings* that control how similar blackboard objects are merged together. Together these mechanisms allow DVMT approximations to be constructed that have significantly reduced durations and well-understood, fairly predictable effects on solution quality (for this work a combination of certainty of beliefs and likelihood of correctness.)

Decker et al. [Decker *et al.*, 1993] provide additional features for the DVMT real-time blackboard application. One new addition is a channelized architecture where each separable part of problem solving has its own *channel* (thread of control) that can be controlled separately from other channels using the parameterized low-level control loop described above. This allows different channels to be running different approximations without interference. Also added is a real-time scheduling component that projects work into the future and tries to avoid overloads by using approximations for tasks and/or postponing parts or all of less important, less time-constrained tasks into the future. This allows the DVMT to achieve predictable real-time performance (including in hard deadline situations), although the issue of the cost of control is not completely addressed.

3.2 Design-to-time real-time scheduling

Design-to-time [Garvey *et al.*, 1993, Garvey and Lesser, 1993] is a generalization of the approximate processing approach where the existence of multiple methods for many tasks is assumed and the problem is to design a solution to a problem that uses all available resources to maximize solution quality within the available time. This work uses a model of computational tasks known as TÆMS [Decker and Lesser, 1993] that models problems as consisting of independent *task groups* that contain possibly dependent tasks. The task/subtask relationship among tasks within a task group forms a directed acyclic graph and is used to calculate the quality of a task (i.e, the quality of a task is a function of the qualities of its subtasks.) At the leaves of this graph are executable methods, which represent actual computations that can be performed by the system. Besides task/subtask relationships, tasks may also have other interdependencies with other tasks in their task group (e.g., the execution of one method enabling

the execution of another, or the use of a rough approximation by one method negatively affecting (hindering) the performance of a method that uses its result.) These interdependencies can have a quantitative effect on the quality and duration of affected methods. This is similar to the effect of the quality of the inputs on performance profiles in Zilberstein's compilation of anytime algorithms [Russell and Zilberstein, 1991]. An example of a TÆMS task structure that models a multi-agent sensor interpretation problem is shown in Figure 5.

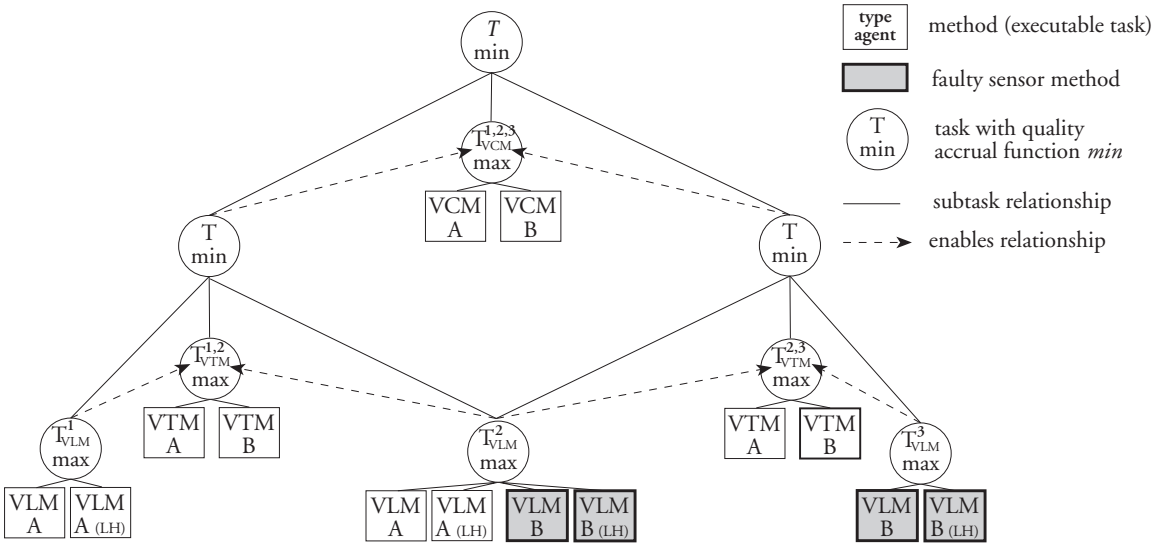

Figure 5: A TÆMS task structure modeling a multi-agent sensor interpretation problem. VLM methods process data for a particular vehicle at a particular location (point in time). VTM methods combine VLM results into tracks. VCM methods complete processing for a vehicle.

The methodology is known as design-to-time because it advocates the use of all available time to generate the best solutions possible. It is a problem-solving method of the type described by D'Ambrosio [D'Ambrosio, 1989] as those that "given a time bound, dynamically construct and execute a problem solving procedure which will (probably) produce a reasonable answer within (approximately) the time available."

Design-to-time can only be successful if the duration and quality associated with methods is fairly predictable. The predictability issue was investigated in detail [Garvey and Lesser, 1993] with the result that the predictability necessary for execution times is based on a complex set of factors that include how heavy the agent's workload is (because in light workload situations it is often acceptable for methods to take longer than expected because slack time is available, whereas in heavy workload situations if a task does not perform as expected it could adversely affect many other tasks) and how difficult it is for the agent to determine when a method is not performing as expected (because if the cost of this monitoring outweighs the potential benefit, then it clearly is not useful). An agent can tolerate uncertainty in its predictions if

- monitoring can be done quickly and accurately, so that when a task will not meet its deadline enough time remains to execute a faster method, or
- intermediate results can be shared among methods, so that when it is necessary to switch to a faster method the intermediate results generated by the previous method can be used, or
- there exists a fast fall back method that quickly generates a minimally acceptable solution.

Garvey and Lesser [Garvey *et al.*, 1993] present an algorithm for finding optimal solutions to a particularly circumscribed design-to-time scheduling problem where:

- The task/subtask relationship forms a tree, rather than a general directed acyclic graph.
- Tasks quality functions are one of *minimum* (AND) or *maximum* (OR).
- Enables relationships may exist among the subtasks of tasks that accumulate quality using minimum. The enables relationships are mutually consistent (i.e., there are no cycles). This corresponds to the situation where there is a body of work that must be completed to satisfy a task and this work must be done in a particular order.
- Hinders relationships may exist in situations where enables may exist and an enabling subtask has a maximum quality accumulation function. In this situation there may be a hinders relationship from the lowest quality method for solving the subtask to the tasks that the subtask enables. This corresponds to the situation where using a crude approximation for a task can have negative effects on the behavior of tasks that use the result of the approximated task.

Figure 6 is an example of such a task structure. The quantitative effects of the hinders relationship are the opposite of those of the *facilitates* relationship [Decker and Lesser, 1993]. Associated with each facilitates/hinders relationship are a pair of parameters that define the power of the effect on the duration and quality of the affected method. Facilitates/hinders increases/decreases the quality of the affected method by a percentage and decreases/increases its duration by a percentage.

These environmental characteristics closely model characteristics seen in a sensor interpretation application. In particular, the enables relationships appear as requirements that low level data be processed before high level interpretations of that data are made, and the hinders relationships appear in the situation where fast, imprecise approximations of low level data processing can both increase the duration and decrease the precision of high level results [Garvey and Lesser, 1993, Lesser and Corkill, 1983].

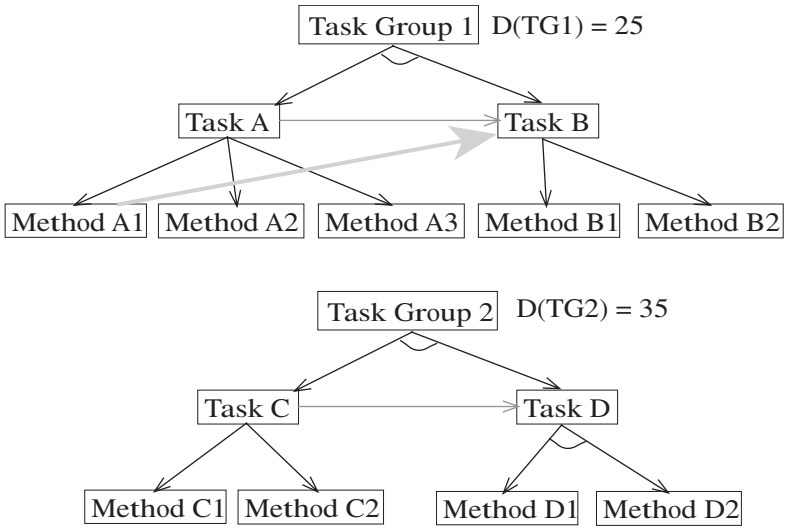

Figure 6: An example task structure consisting of two task groups of the form the optimal design-to-time algorithm can schedule. The dark lines indicate subtask relationships. The thin gray lines represent enables constraints. The thick gray line represents a hinders constraint.

The algorithm essentially involves generating all sets of methods that can solve the problem, pruning those that are superseded by other sets of methods that generate greater or equal quality in equal or less

time. Performance results for the algorithm are given and suggest that the average performance of the algorithm is near linear for fairly small task sets, but that it probably will not scale as task size increases and likelihood of relationships increases. One other interesting result is that the number of schedules to be considered is limited by the number of distinct quality values that methods have (because minimum and maximum do not create new values, but just return one of the values they are given), so the runtime of the scheduler can be reduced by reducing the number of distinct quality values, for example, by clustering similar values together. This tradeoff between schedule precision and scheduler runtime has a contract anytime algorithm character.

This algorithm was developed and tested using TÆMS [Decker and Lesser, 1993], allowing it to be studied both analytically and in a simulation environment. More recent design-to-time research has focused on heuristic scheduling techniques and on the interface between a design-to-time scheduler and the application that it is embedded in. For another perspective on the problem of real-time scheduling of AI tasks see Stankovic et al. [Stankovic *et al.*, 1989].

3.3 Time-constrained reasoning under uncertainty

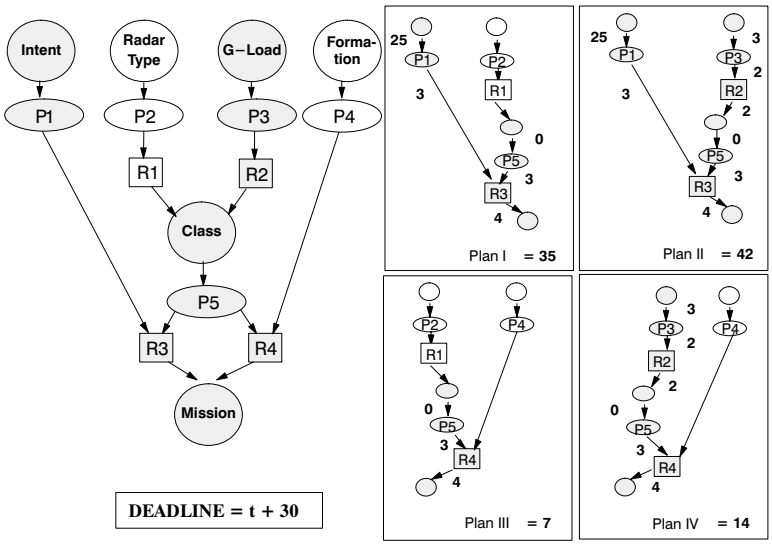

Figure 7: An example of a set of plans from Bonissone and Halverson's system. One the left is the entire set of plans; on the right are four individual plans with different runtimes and expected values.

The work of Bonissone and Halverson [Bonissone and Halverson, 1990] looks at solving *dynamic classification problems* which are classification problems in which the rate of change of the environment is as fast as the time to do a diagnosis—meaning that some of the data on which a diagnosis is based might have changed by the time the diagnosis is completed. Their approach involves generating a set of plans at compile time by pre-evaluating plan usefulness and caching this information. Figure 7 shows an example of a set of plans. At runtime they select some or all of the plans that can handle a particular query, based on how much time is available to answer the query. These plans can be thought of as multiple methods for satisfying a particular goal. Their control mechanism executes plans in the order of expected utility and may opportunistically change the order based on the actual values produced by earlier plans. They assume that higher priority queries may interrupt execution on a given query, and this is taken into account when deciding in what order to execute plans. This emphasis on doing as much as possible at design time recurs in real-time AI research (as in systems-oriented real-time research), especially in work that emphasizes actual applications.

3.4 Etzioni's marginal utility heuristic

Etzioni [Etzioni, 1991] looks at the control problem for a time-constrained agent in economic terms. When the use of a resource (such as time) for one action precludes that resource being used by another action, the executed action is said to have an *opportunity cost*. When several actions contend for the same resource, each action has an opportunity cost that is the maximum of the utilities of the other actions. *Marginal utility* is defined as the derivative of the utility function with respect to a cost variable (e.g., time). Etzioni examines the usefulness of what he calls the MU heuristic, which says to always execute the action that has the highest marginal utility[4].

The particular control problem that Etzioni focuses on is one in which there are a number of independent goals, each with some number of associated methods. Each of these methods has expected utility and runtime values. The utility of a method is related to how well the method is expected to satisfy the goal, as well as how likely the method is to satisfy the goal. The control problem is to maximize the total additive value returned by satisfying the goals. He shows that this problem is NP-hard and for that reason he uses his MU heuristic to determine which actions to execute. This reduces the complexity of the control problem to linear (or even sublinear per action decision if you assume that the utility and runtime of an action do not change due to the execution of other actions).

One interesting aspect of Etzioni's work is that he doesn't assume that completely accurate utility and runtime values are available initially for all actions in all states. He describes a learning mechanism that calculates expected means for utility and runtime through repeated executions of the system. When the variance in these expected means for an action is too high his system uses an ID3-like mechanism to find some attribute distinguishing sets of states in which the expected utility and runtime means differ significantly. He uses this information to construct a *classification tree* (also known as a *decision tree*) of utility and runtime means for each goal type. Then, the next time a goal of that type appears he traverses the tree, deciding which branch to follow based on current attributes of the environment. This means that his estimates of utility and runtime will become increasingly accurate as problem solving progresses. Of course, in many real-time situations it may not be acceptable to not perform adequately in real-time initially and gradually learn to do so at runtime. In these situations the learning mechanism could be used at design-time to construct a system that performs adequately in real-time.

3.5 ABE/RT architecture

ABE [Lark *et al.*, 1990] is essentially a CASE tool for building software applications with major AI components. ABE has several *frameworks* that support different ways of combining component modules to build applications. Often these component modules are themselves built in ABE, possibly using many different frameworks. At the base are so-called *black box* modules that are built in regular programming languages (Common Lisp, C, etc. . .).

ABE has a real-time framework (ABE/RT) that allows users to build real-time applications (e.g., a prototype of the Pilot's Associate was built in the ABE/RT framework [Lark *et al.*, 1990]). The ABE/RT framework differs from other ABE frameworks in that it requires information about the runtime of modules, not just their input/output behavior. ABE/RT allows modules to be viewed from three perspectives. One perspective describes the flow of events through the module. Events may be sent to many internal modules; the only restriction is that the flow of events be acyclic. A second perspective describes what are known as *event processing plans* (EPPs). EPPs are particular paths of event flow

[4]This heuristic is closely related to Boddy and Dean's [Boddy, 1991] scheduling method that gives runtime to the anytime algorithm with the highest gradient in its expected performance profile. Both are what Russell calls *meta-greedy algorithms* [Russell and Wefald, 1991] i.e., they choose the action appearing to have the highest immediate benefit.

through a module. Each EPP can be thought of as one of a set of methods for responding to a particular input event. Associated with an EPP is detailed information about its resource requirements, including its worst-case runtime. ABE/RT provides tools to help designers determine these requirements, but most of this is done empirically rather than through analysis. The third perspective describes what resources to give to which EPPs in particular circumstances. One part of the definition of an ABE/RT application is a set of *modes* of the system. Modes define the priorities of the system and change based on perceived changes in the environmental situation. For example, a system that assists a pilot in flight might have different modes for different parts of a flight scenario (take-off, ascending, descending, landing) and for different contingencies in flight (one engine out, enemy pursuing, ...). The third ABE/RT perspective describes how to allocate resources among EPPs for each mode of the system. That is, this third perspective statically allocates resources to particular methods in a particular mode under maximum event arrival assumptions.

The intended usage of the ABE/RT framework is to build applications that can respond to real-time events, reacting differently depending on the dynamically changing priorities associated with the a particular situation. Presumably several different EPPs are available to respond to a particular event and which one is chosen depends on the mode of the system. These EPPs differ in the way they solve the problem, the assumptions they make about which component systems are available, and the resources they need to execute. Constructing the mapping that describes which EPPs are used in which mode is a multiple method scheduling problem that is solved statically by hand at design time (with the assistance of online tools) rather than dynamically by a computational component. This decision was made because it was assumed that there would not be enough time at runtime to solve the problem, and because of a need to show that an acceptable solution is always possible.

4 System Architectures

One research direction in real-time AI has been in building large applications or architectures that embody real-time concerns in many components. With all of these projects the eventual goal is to have overall real-time performance through several interacting components, although often not all of these components have initially been concerned with real-time performance. Often these large systems combine fast-acting reactive components with more deliberative cognitive components.

System architectures to be discussed include Guardian [Hayes-Roth, 1990, Hayes-Roth *et al.*, 1992, Washington and Hayes-Roth, 1989], Phoenix [Howe *et al.*, 1990], PRS [Ingrand and Georgeff, 1990, Ingrand *et al.*, 1992] and CIRCA [Musliner *et al.*, 1993]. Both the ABE/RT system and the real-time DVMT architecture as discussed in detail above fit into this category as well. Although we do not discuss it in detail here, Brooks' work on the subsumption architecture [Brooks, 1986] (which is generally considered to be reactive) can be thought of as a layered architecture where the higher layers are deliberative and the lower layers are reactive, which makes it clear that the exact distinctions between reactive and deliberative are somewhat fuzzy and hard to define.

4.1 Guardian application

Guardian [Hayes-Roth, 1990, Hayes-Roth *et al.*, 1992, Washington and Hayes-Roth, 1989] is a medical monitoring application built in a blackboard architecture (BB*). Real-time aspects of this work include a separate input manager that filters and processes inputs [Washington and Hayes-Roth, 1989], a satisficing control cycle to bound the amount of time spent doing metalevel reasoning [Collinot and Hayes-Roth, 1990, Hayes-Roth and Collinot, 1991], and an anytime diagnosis component [Ash and Hayes-Roth, 1993]. Figure 8 shows an overview of the agent architecture used by Guardian.

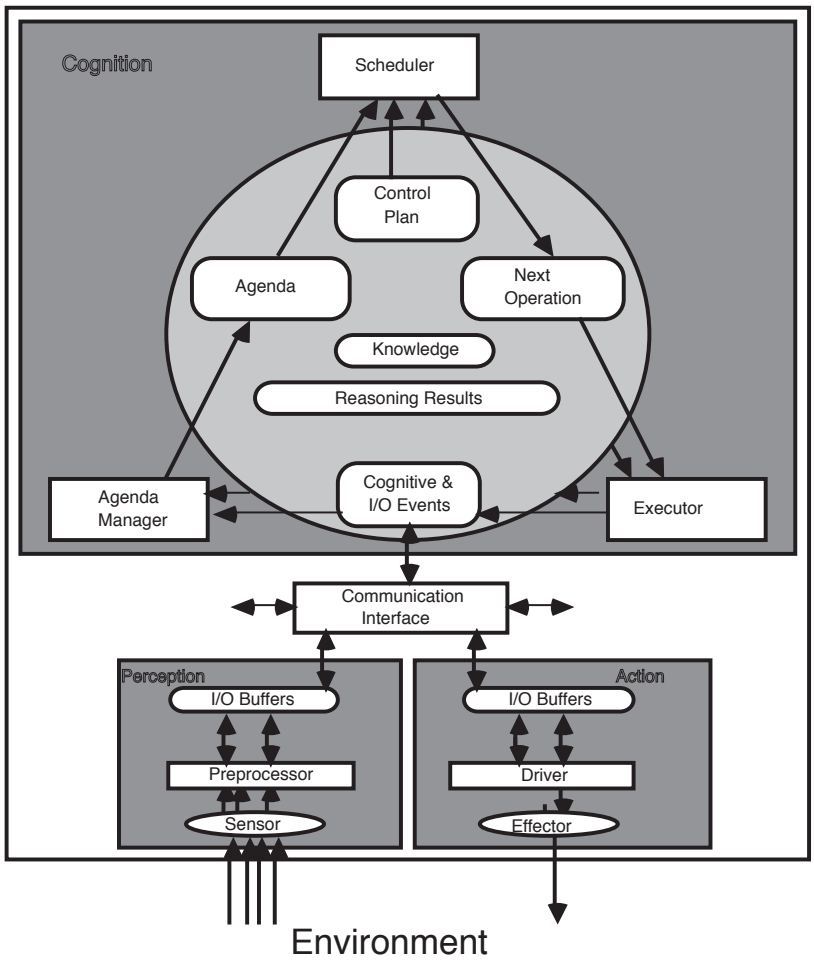

Figure 8: The agent architecture used by Guardian. Curved boxes represent blackboard data structures. Rectangular boxes represent concurrently executing processes. Information flow is indicated by the directional arrows.

In the Guardian medical monitoring application, large amounts of input data arrive at the system. Much of this is low level data that just confirms current patterns, but occasionally important or unexpected information arrives. Filtering of this data is necessary, but this filtering needs to be dynamic and intelligent to avoid both overburdening the cognitive component with needless detail and not informing it of important new information. Washington and Hayes-Roth [Washington and Hayes-Roth, 1989] describe how Guardian's cognitive component dynamically builds and modifies filters to perform this task as problem solving progresses. The input data is processed on a separate I/O processor that filters the data using the dynamically-defined filters as controlled by the cognitive component. Thus the cognitive component is able to dynamically control its I/O load. They emphasize that such dynamic construction is necessary because of the changing requirements of the filters in different problem solving situations.

The satisficing control cycle and anytime diagnosis component are discussed in detail in Sections 2.3.3 and 2.3.4, respectively. Together these components give Guardian the ability to be reactive in situations that require it, by using input filters to separate out important data, and a satisficing control cycle to quickly determine how to respond to it. As work on Guardian has progressed, more and more components of the system have achieved real-time performance. Guardian is a good example of a system that combines several of the techniques described in this paper (satisficing control, anytime algorithms, dynamic filtering) in appropriate components of the system to achieve overall real-time performance.

4.2 Real-time work in Phoenix

The Phoenix project [Howe *et al.*, 1990] is investigating the design of autonomous agents. The two main components of the project are a *simulator* that simulates the spread of forest fires and provides tools to allow the fires to be fought, and an *agent architecture* that is used to instantiate agents that are tested in the simulated environment. The main real-time related contribution of Phoenix is in its examination of the necessity for a reactive component in an otherwise deliberative AI system. In essence, the reactive component reacts to immediate problems in the environment (in the Phoenix case this is nearby fires, changes in terrain, ...) and the cognitive component plans to solve the high-level problems presented to the agent (such as how to put out a particular fire, what path to take from one point to another, ...). The basic idea of having reactive and cognitive components within an agent architecture is one way that AI systems can work effectively in domains with soft real-time constraints.

4.3 PRS architecture

The Procedural Reasoning System (PRS) [Ingrand and Georgeff, 1990, Ingrand *et al.*, 1992] is an architecture for embedded real-time systems that need to deliberate in real-time. A PRS agent consists of a database containing the systems current beliefs, a set of current goals, a library of plans (called *knowledge areas* KAs) that describe sequences of actions and tests that can be performed to meet a goal or react to a situation, and an intention structure that consists of a partially ordered set of those plans chosen for execution. An interpreter works with these components to select an appropriate plan (KA) based on beliefs and goals, place that plan in the intention structure and execute it. Metalevel KAs are used to decide among multiple applicable domain KAs in a particular situation, reason about the failure to satisfy goals, and manage the flow of control among intentions (including determining when to continue applying metalevel KAs versus executing the current domain-level plan). KAs are interruptible when external events cause changes to the database, thus allowing rapid response to changing environmental situations. Ingrand and Georgeff [Ingrand and Georgeff, 1990] describe the results of experiments that look at the response time of a PRS system to external events (e.g., the time from when an event arrives at the system until the system has decided how to respond to it). They show that PRS can be configured to *notice* (begin to respond to) world events within a bounded amount of time. At this time PRS cannot

guarantee to *respond* to events within a bounded time, because it does not completely ignore other incoming events. These results suggest that PRS might be useful, at least for soft real-time applications.

4.4 CIRCA architecture

CIRCA [Musliner *et al.*, 1993] is an architecture intended to support applications that combine real-time and AI components. At a high level its is very similar to the Phoenix approach. In CIRCA there is a real-time component (corresponding to the Phoenix reactive component) and an AI component (corresponding to the Phoenix cognitive component.) The CIRCA work assumes that AI software is inherently unpredictable, so no assumptions can be made about its performance. The real-time component repeatedly executes a periodic schedule consisting of simple test-action pairs (called TAPs). TAPs are intended both to achieve short-term goals and to maintain the system in a safe state. These TAP schedules guarantee that each TAP will receive up to its worst-case execution time (using traditional cyclic scheduling techniques). In that sense the system "guarantees" real-time performance. There is no guarantee that the TAPs will always satisfy all system goals, only that the TAPs will always be run when they are scheduled. It is the responsibility of the AI component to generate TAP schedules to respond to the current problem-solving situation. In a sense the AI component is dynamically constructing simple reactive systems to satisfy system goals in the current problem solving situation. Hendler and Agrawala [Hendler and Agrawala, 1990] have done similar work on combining a real-time reactive component with higher level AI planning, all implemented in the MARUTI real-time operating system.

5 Summary of research directions

The dual goals of this survey have been to show the progress that is being made in the field of real-time AI and to illustrate the fundamental tradeoffs that are necessary because of inherent limitations imposed by time constraints. This section summarizes our thoughts on these conflicting drives within real-time AI and suggests possible future research directions for the field.

5.1 Recent progress in real-time AI

One major area of progress has been in the integration of real-time components for all aspects of a problem-solving system. Early work in real-time AI focused on defining and elucidating particular useful techniques (e.g., anytime algorithms). These techniques are the backbone on which real-time AI must be built, but on their own they are not very useful for building actual real-time applications. More recently work has focused on ways to combine these techniques together to build larger systems. Research has focused on homogeneously combining low level instances of a technique (e.g., compilation of anytime algorithms and design-to-time scheduling of multiple methods) and on heterogeneously using different techniques in different components of a larger application (e.g., combining anytime algorithms with satisficing control and filtering in Guardian).

Another area of progress has been in building real-time AI components intended to fit into larger real-time application systems. Some initial work in real-time AI explored techniques for solving particular problems in real-time, but did not worry about embedding those solutions in larger applications. Embedding of this form requires AI systems that allow their behavior to be guided by outside influences. Examples of systems that do this are the CIRCA and Phoenix systems where the cognitive AI component works along with a reactive component to achieve overall real-time performance. Clearly large, practical real-time systems that work in dynamic domains are going to benefit from some kind of cognitive component that can respond to unexpected situations, even in critical hard real-time situations where

at least some real-time behavior will probably require guarantees made using standard systems-oriented real-time techniques.

An area where progress is beginning to be made is in the integrating of anytime algorithm and multiple method approaches to create hybrid systems. For example, the compilation techniques of Zilberstein [Russell and Zilberstein, 1991] can be used to compile programs consisting of both anytime and traditional algorithms (where the performance profile of a traditional algorithm is presumably a single step function). Alternatively, an anytime algorithm can be thought of as multiple methods where a small set of discrete runtime allocations with their expected qualities (according to the performance profile) define a set of methods that can be scheduled using, for example, design-to-time scheduling techniques. Such hybrid systems could combine the best features of both approaches and avoid trying to unnaturally fit algorithms of one type into the requirements of the other approach.

Progress has also been made in moving cognitive reasoning from compile/design time to runtime, while still maintaining overall real-time performance. From the beginning of real-time AI research, at least some of the work has focused on making dynamic, runtime decisions, but often this work ignored the cost of control reasoning. More recently work has focused on either limiting the cost of control by using simple algorithms with known worst-case performance (e.g., Etzioni's MU heuristic), or using sophisticated meta-reasoning to make tradeoffs in the cost of continued control reasoning versus performing the current best action (e.g., Horvitz's work on partitioning resources among metareasoning and domain tasks).

5.2 Problem and method complexity limitations used in real-time AI

In order to solve difficult problems in real-time it is necessary to place some limitations on the complexity of problem-solving. One way to consider the different real-time AI approaches described above is to examine how they have limited the complexity of the problem in some way so as to allow reasonable system performance. These complexity limitations can be in the representation of the problem to be solved and/or in the solution methods used by the approach.

One possible definition of real-time AI is that it involves attempting to build real-time systems that solve problems normally thought to require intelligence. Often these problems have some characteristics (such as extensive use of search) that make solving them using traditional real-time techniques difficult if not impossible (e.g., because of very large worst-case response times.) Given that, how can real-time AI systems continue to get away with making so many complexity-reducing assumptions? The answer is that without some limitations on the complexity of problem-solving, no real-time solution is possible (which is why systems-oriented real-time researchers have mostly avoided these kinds of problems.) The long term goal is to understand how each of these complexity limitations can be avoided (possibly by making complexity-simplifying assumptions in other parts of the problem) and what is potentially gained by doing so. It is understanding the tradeoffs among different complexity limitations (i.e., what is lost in one dimension while something is gained in another) that motivate much of this work.

Complexity limitations explored in real-time AI systems fall into a few broad categories including limitations in the structure of the computation, the criteria for solution acceptability, the relative costs of control versus domain reasoning, and the predictability of tasks. More specifically, complexity limitation categories include:

- **structure of the computation**—This approach to reducing complexity involves limiting the kinds of computations that the system can perform or making assumptions about the forms these computations take. Note that because of the reduced complexity of the problem representation, it is sometimes possible to make optimal decisions in the constrained problems. Examples include:
  - **task independence**—Many real-time AI systems assume that individual tasks are indepen-

dent, i.e., how one task is solved has no effect on possible solutions to other tasks, except in reducing available time to solve the other tasks. Assumptions of task independence generally significantly reduce the complexity of solution algorithms, usually by allowing separate subproblems to be solved independently without worrying about interactions. Examples of work that avoid this limitation include systems-oriented work on scheduling that incorporates resources [Zhao *et al.*, 1987], compilation of anytime algorithms [Zilberstein, 1993], and design-to-time scheduling [Garvey *et al.*, 1993].

  - **anytime algorithms**—Some work in both systems-oriented [Liu *et al.*, 1991b] and AI real-time [Dean and Boddy, 1988, Russell and Zilberstein, 1991] assumes that many or all problems are to be solved using anytime algorithms (i.e., algorithms that always have an answer at hand and provide higher quality answers if they are given more time up to some maximum quality). It is probable that not all problems have anytime algorithm solutions. The expected advantages of anytime algorithms are that they respond gracefully to unpredictability and provide acceptable solutions quickly in time-constrained situations. One step in the direction of avoiding the construction problem is Zilberstein's work on compiling anytime algorithms [Zilberstein, 1993] that shows how composite anytime algorithms can be constructed from simpler anytime algorithms.
  - **soft deadlines**—Some of the work in real-time AI assumes that deadlines are soft (i.e., it is okay to finish work after a deadline), and that missing any particular deadline is not catastrophic (i.e., some tasks with deadlines can be ignored completely). One example of this is systems that *on average* meet deadlines, while not making any guarantees about any particular deadline. Most systems-oriented real-time research does not have this limitation. This is one of the dimensions that real-time AI often compromises to try to gain advantages elsewhere.

- **criteria for solution acceptability**—Much work in real-time AI makes the assumption that satisficing rather than optimal solutions are acceptable in many situations. Satisficing solutions can be approximate in precision, completeness and/or certainty. Often satisficing takes the form of using heuristic problem solving techniques that may not always find an optimal solution, such as the TSP algorithm shown in Figure 1. Satisficing is useful only if nonoptimal, acceptable solutions exist for subproblems (which might not always be the case, for example in yes/no situations) and the use of a satisficing solution for some subproblems does not cause other subproblems to increase in difficulty and thus overwhelm the benefit.

- **relative cost of control versus domain reasoning**—Much of the work in real-time AI assumes that the cost of control is negligible and can safely be ignored. This assumption can be justified if the cost of control is a predictable polynomial function of the size of the problem and the grain-size of control subproblems is small relative to domain subproblems. This has the effect of limiting the kinds of applications to which the techniques can be applied. More recently, some work in real-time AI has avoided making this limiting assumption by explicitly taking real-time constraints into account in a real-time control component [Hayes-Roth *et al.*, 1992] or by explicitly reasoning about the expected value of control versus domain reasoning [Horvitz and Breese, 1990, Ingrand and Georgeff, 1990, Russell and Wefald, 1991].

- **predictability of tasks**—All real-time work has to make some assumptions about the accuracy and completeness of information about tasks to be performed. Some work assumes that information is predictable enough to allow static (sometimes even design-time) solutions to be generated. This has the advantage of avoiding excessive runtime control costs, at the expense of either

occasionally using inaccurate predictions (and thus not performing as expected) or using worst-case performance information and not using resources very efficiently. Even dynamic, runtime problem solvers have to make assumptions about the predictability of task information. One approach to this problem is to monitor the performance of tasks and adjust behavior when performance is not as expected [Garvey and Lesser, 1993]. Anytime algorithms make assumptions about the accuracy of performance profile information, but, in general, they should perform better in unpredictable situations, because they always have an answer at hand. Note that even in situations where accurate and complete information is available, uncertainty due to problem complexity may prevent static solutions (e.g., chess).

Another real-time AI solution to the requirement for agents that respond dynamically to a changing environment has been to build *reactive AI systems* [Agre and Chapman, 1987, Firby, 1987] that do not deliberate at all, but react directly to inputs. While this survey focuses on deliberative real-time AI, at some level of required reaction time, the only reasonable solution method is reactive. Some work in deliberative real-time AI has looked at having a reactive component in an otherwise deliberative system [Howe *et al.*, 1990, Musliner *et al.*, 1993]. Another approach to this problem is to assume that the real-time AI system is just a component that is embedded in a larger application system, and that the larger application will only pass appropriate problems on to the AI component.

As this survey shows, real-time AI has made significant progress in the last few years, but it still has a lot of work to do in systematically examining the costs, benefits and tradeoffs of different complexity-reducing assumptions. Another important area for future research is the integration of different approaches to take advantage of strengths and minimize the effects of weaknesses. This could take the form of a layered architecture that uses, for example, a hard real-time scheduler at the base and AI reasoning techniques at higher levels, or a more component-oriented architecture that uses different approaches in different situations, for example, depending on the predictability of tasks and the tightness of deadlines.

Acknowledgments

We would like to thank Shlomo Zilberstein, Piero Bonissone, Peter Halverson, Barbara Hayes-Roth and Keith Decker for the use of figures that appear in this paper.

References

[Agre and Chapman, 1987] Philip E. Agre and David Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 268–272, Seattle, WA, July 1987.

[Ash and Hayes-Roth, 1993] David Ash and Barbara Hayes-Roth. A comparison of action-based hierarchies and decision trees for real-time performance. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 568–573, Washington, D.C., July 1993.

[Ash *et al.*, 1993] D. Ash, G. Gold, A. Seiver, and B. Hayes-Roth. Guaranteeing real-time response with limited resources. *Artificial Intelligence in Medicine*, 5:49–66, 1993.

[Boddy and Dean, 1989] Mark Boddy and Thomas Dean. Solving time-dependent planning problems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989.

[Boddy and Dean, 1993] Mark Boddy and Thomas Dean. Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 1993. To appear.

[Boddy, 1991] Mark Boddy. Solving time-dependent problems: A decision-theoretic approach to planning in dynamic environments. Ph.D. Dissertation CS-91-06, Department of Computer Science, Brown University, Providence, RI, 1991.

[Bonissone and Halverson, 1990] Piero P. Bonissone and Peter C. Halverson. Time-constrained reasoning under uncertainty. *The Journal of Real-Time Systems*, 2(1/2):25–45, 1990.

[Brooks, 1986] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, March 1986.

[Chung *et al.*, 1990] J. Y. Chung, J. W. S. Liu, and K. J. Lin. Scheduling periodic jobs that allow imprecise results. *IEEE Transactions on Computers*, 39:1156–1173, 1990.

[Collinot and Hayes-Roth, 1990] Anne Collinot and Barbara Hayes-Roth. Real-time control of reasoning: Experiments with two control models. In *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 263–270, November 1990.

[Cooper, 1990] G. F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42(2/3):393, 1990.

[D'Ambrosio, 1989] Bruce D'Ambrosio. Resource bounded-agents in an uncertain world. In *Proceedings of the Workshop on Real-Time Artificial Intelligence Problems*, IJCAI-89, Detroit, August 1989.

[Dean and Boddy, 1988] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 49–54, St. Paul, Minnesota, August 1988.

[Dean and Wellman, 1991] Thomas Dean and Michael Wellman. *Planning and Control*. Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[Decker and Lesser, 1993] Keith S. Decker and Victor R. Lesser. Quantitative modeling of complex computational task environments. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 217–224, Washington, July 1993.

[Decker *et al.*, 1990] Keith S. Decker, Victor R. Lesser, and Robert C. Whitehair. Extending a blackboard architecture for approximate processing. *The Journal of Real-Time Systems*, 2(1/2):47–79, 1990.

[Decker *et al.*, 1993] Keith S. Decker, Alan J. Garvey, Marty A. Humphrey, and Victor R. Lesser. A real-time control architecture for an approximate processing blackboard system. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(2):265–284, 1993.

[Dey *et al.*, 1993] J.K. Dey, James Kurose, and Don Towsley. On-line processor scheduling for a class of IRIS (increasing reward with increasing time) real-time tasks. CS Technical Report 93–09, University of Massachusetts, 1993.

[Etzioni, 1991] Oren Etzioni. Embedding decision-analytic control in a learning architecture. *Artificial Intelligence*, 49:129–159, 1991.

[Firby, 1987] R. James Firby. An investigation into reactive planning in complex domains. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 202–206, Seattle, WA, July 1987.

[Garvey and Lesser, 1993] Alan Garvey and Victor Lesser. Design-to-time real-time scheduling. *IEEE Transactions on Systems, Man and Cybernetics*, 23(6), 1993. To appear.

[Garvey *et al.*, 1993] Alan Garvey, Marty Humphrey, and Victor Lesser. Task interdependencies in design-to-time real-time scheduling. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 580–585, Washington, D.C., July 1993.

[Hayes-Roth and Collinot, 1991] Barbara Hayes-Roth and Anne Collinot. Scalability of real-time reasoning in intelligent agents. Technical Report KSL 91-08, Knowledge Systems Laboratory, Stanford University, 1991.

[Hayes-Roth *et al.*, 1992] B. Hayes-Roth, R. Washington, D. Ash, A. Collinot, A. Vina, and A. Seiver. Guardian: A prototype intensive-care monitoring agent. *Artificial Intelligence in Medicine*, 4:165–185, 1992.

[Hayes-Roth, 1990] Barbara Hayes-Roth. Architectural foundations for real-time performance in intelligent agents. *The Journal of Real-Time Systems*, 2(1/2):99–125, 1990.

[Hendler and Agrawala, 1990] James Hendler and Ashok Agrawala. Mission critical planning: AI on the MARUTI real-time operating system. In *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 77–84, November 1990.

[Ho *et al.*, 1992] Kevin I-J. Ho, Joseph Y-T. Leung, and W-D. Wei. Scheduling imprecise computation tasks with 0/1-constraint. Technical Report UNL–CSE–92–16, University of Nebraska-Lincoln, 1992.

[Horvitz and Breese, 1990] Eric J. Horvitz and John S. Breese. Ideal partition of resources for metareasoning. Technical report KSL-90-26, Knowledge Systems Laboratory, Stanford University, March 1990.

[Horvitz and Rutledge, 1991] Eric J. Horvitz and Geoffrey Rutledge. Time-dependent utility and action under uncertainty. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, Los Angeles, CA, July 1991.

[Horvitz *et al.*, 1989] Eric J. Horvitz, Gregory F. Cooper, and David E. Heckerman. Reflection and action under scarce resources: Theoretical principles and empirical study. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1121–1127, Detroit, MI, August 1989.

[Horvitz, 1988] Eric J. Horvitz. Reasoning under varying and uncertain resource constraints. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 111–116, St. Paul, MN, August 1988.

[Horvitz, 1989] Eric J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In L. N. Kanal, T. S. Levitt, and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence 3*. Elsevier Science Publishers, 1989.

[Howe *et al.*, 1990] Adele E. Howe, David M. Hart, and Paul R. Cohen. Addressing real-time constraints in the design of autonomous agents. *The Journal of Real-Time Systems*, 2(1/2):81–97, 1990.

[Ingrand and Georgeff, 1990] F. F. Ingrand and M. P. Georgeff. Managing deliberation and reasoning in real-time AI systems. In *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 284–291, November 1990.

[Ingrand *et al.*, 1992] Francois F. Ingrand, Michael P. Georgeff, and Anand S. Rao. An architecture for real-time reasoning and system control. *IEEE Expert*, pages 34–44, December 1992.

[Jensen *et al.*, 1985] E. Douglas Jensen, C. Douglass Locke, and Hideyuki Tokuda. A time-driven scheduling model for real-time operating systems. In *Proceedings of the 1985 Real-time Systems Symposium*, pages 112–122, December 1985.

[Kenny and Lin, 1991] Kevin B. Kenny and Kwei-Jay Lin. Building flexible real-time systems using the Flex language. *IEEE Computer*, 24(5):70–78, May 1991.

[Korf, 1990] Richard E. Korf. Depth-limited search for real-time problem solving. *The Journal of Real-Time Systems*, 2(1/2):7–24, 1990.

[Lark *et al.*, 1990] Jay S. Lark, Lee D. Erman, Stephanie Forrest, Kim P. Gostelow, Frederick Hayes-Roth, and David M. Smith. Concepts, methods, and languages for building timely intelligent systems. *The Journal of Real-Time Systems*, 2(1/2):127–148, 1990.

[Lesser and Corkill, 1983] Victor R. Lesser and Daniel D. Corkill. The distributed vehicle monitoring testbed. *AI Magazine*, 4(3):63–109, Fall 1983.

[Lesser *et al.*, 1988] Victor R. Lesser, Jasmina Pavlin, and Edmund Durfee. Approximate processing in real-time problem solving. *AI Magazine*, 9(1):49–61, Spring 1988.

[Leung *et al.*, 1992] Joseph Y-T. Leung, Vincent K.M. Yu, and W-D. Wei. Minimizing the weighted number of tardy task units. Technical report, University of Nebraska-Lincoln, 1992.

[Liu *et al.*, 1991a] J. W. S. Liu, K. J. Lin, W. K. Shih, A. C. Yu, J. Y. Chung, and W. Zhao. Algorithms for scheduling imprecise computations. In Andr'e M. van Tilborg and Gary M. Koob, editors, *Foundations of Real-Time Computing: Scheduling and Resource Management*. Kluwer Academic Publishers, 1991.

[Liu *et al.*, 1991b] J. W. S. Liu, K. J. Lin, W. K. Shih, A. C. Yu, J. Y. Chung, and W. Zhao. Algorithms for scheduling imprecise computations. *IEEE Computer*, 24(5):58–68, May 1991.

[Marlin *et al.*, 1990] Chris Marlin, Wei Zhao, Graeme Doherty, and Andrew Bohonis. GARTL: A real-time programming language based on multi-version computation. In *Proceedings of the International Conference on Computer Languages*, pages 107–115, New Orleans, LA, March 1990.

[Musliner *et al.*, 1993] David J. Musliner, Edmund H. Durfee, and Kang G. Shin. CIRCA: A cooperative intelligent real-time control architecture. *IEEE Transactions on Systems, Man and Cybernetics*, 23(6), 1993. To appear.

[Russell and Wefald, 1991] Stuart Russell and Eric Wefald. *Do the Right Thing: Studies in Limited Rationality*. MIT Press, Cambridge, MA, 1991.

[Russell and Zilberstein, 1991] Stuart J. Russell and Shlomo Zilberstein. Composing real-time systems. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 212–217, Sydney, Australia, August 1991.

[Shih *et al.*, 1991] Wei-Kuan Shih, Jane W. S. Liu, and Jen-Yao Chung. Algorithms for scheduling imprecise computations with timing constraints. *SIAM Journal on Computing*, 20(3):537–552, June 1991.

[Stankovic *et al.*, 1989] J. A. Stankovic, K. Ramamritham, and D. Niehaus. On using the Spring kernel to support real-time AI applications. In *Proceedings of the EuroMicro Workshop on Real-time Systems*, 1989.

[Strosnider and Paul, 1993] Jay K. Strosnider and C. J. Paul. A structured view of real-time problem solving. Technical report, Carnegie Mellon University, Department of Electrical and Computer Engineering, 1993.

[Washington and Hayes-Roth, 1989] Richard Washington and Barbara Hayes-Roth. Input data management in real-time AI systems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 250–255, Detroit, MI, August 1989.

[Zhao *et al.*, 1987] W. Zhao, K. Ramamritham, and J. A. Stankovic. Scheduling tasks with resource requirements in hard real-time systems. *IEEE Transactions on Software Engineering*, May 1987.

[Zilberstein and Russell, 1992a] Shlomo Zilberstein and Stuart J. Russell. Constructing utility-driven real-time systems using anytime algorithms. In *Proceedings of the IEEE Workshop on Imprecise and Approximate Computation*, pages 6–10, Phoenix, AZ, December 1992.

[Zilberstein and Russell, 1992b] Shlomo Zilberstein and Stuart J. Russell. Efficient resource-bounded reasoning in AT-RALPH. In *Proceedings of the First International Conference on AI Planning Systems*, College Park, Maryland, June 1992.

[Zilberstein, 1993] Shlomo Zilberstein. Operational rationality through compilation of anytime algorithms. Ph.D. Dissertation, Department of Computer Science, University of California at Berkeley, Berkeley, CA, 1993.