Any-Dimension Algorithms and Real-Time AI

David J. Musliner Edmund H. Durfee Kang G. Shin

Computer Science and Engineering Division
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109-2122

{djm,durfee,kgshin}@eecs.umich.edu
(313) 936-2495

ABSTRACT

In this paper, we extend the concepts underlying the any-time computation paradigm to other measures of resource usage or output quality. The resulting notion of *any-dimension algorithms* forms a generally useful approach to bounded-resource computations. We discuss features and extensions of any-dimension algorithms that allow them to provide guarantees on more than one performance dimension. We then show how the any-dimension paradigm provides a concise expression of the difficulties inherent in real-time AI systems. This viewpoint clarifies the tradeoffs that must be made to achieve the conflicting goals of real-time and AI methods, and we briefly describe the architectural approach we are developing to make these tradeoffs.

The work reported in this paper was supported in part by the National Science Foundation under Grants IRI-9209031 and IRI-9158473, and by a NSF Graduate Fellowship. The opinions, findings, and recommendations expressed in this publication are those of the authors, and do not necessarily reflect the views of the NSF.

1 Introduction

As we rely more heavily on computing systems, the inherent limitations of these systems become extremely significant. For example, all computing systems operate with bounded resources, including limited memory, communication bandwidth, and computation speed. A bounded computing system that must perform a significant task where failure to act in time can lead to costly damage or loss of life is called a *hard real-time system.* Typically, a hard real-time system is employed to control some activity or process, such as monitoring a nuclear power plant or controlling the actuators on an advanced aircraft.

When applying a computer control system to a hard real-time domain, the system designer must ensure that the resource-bounded system will meet the demands of its environment [14]. Researchers have developed two classes of methods for obtaining performance guarantees given a limited set of system resources. In the most common "strategic" approach, a scheduling mechanism is given information about resource availability and future computational tasks, and determines how to execute those tasks in order to avoid resource conflicts and optimize some measure of performance. This approach is well-suited to simple control algorithms and static domains, where resource needs and availability are predictable, so that the resulting schedule of tasks can be followed precisely. However, as more complex AI methods are applied to control problems in dynamic, unpredictable domains, strategic methods fall short because their scheduling algorithms cannot keep up with changing resource needs and availability. In addition, the high-variance, search-based AI methods used in complex control systems are problematic for strategic schedulers because scheduling of these AI methods based on worst-case performance predictions can lead to severe underutilization of system resources [12].

To achieve intelligent real-time control in dynamic domains, "tactical" resource management approaches have been developed that avoid *a priori* scheduling, relying instead on the computational tasks themselves to manage their resource usage. These tactical methods are exemplified by any-time algorithms [2, 6, 13], which can be interrupted at any time to yield a result, possibly with reduced precision, confidence, or completeness. Any-time algorithms provide an on-line, dynamic method for guaranteeing the timeliness of a result, but the quality of the result may not be acceptable.

In this paper, we show that any-time algorithms are just one particular approach in a larger class of tactical algorithms that can provide performance guarantees along dimensions describing both resource usage and solution quality. We characterize these *any-dimension* algorithms in the next two sections, and we show in Section 4 that guaranteeing combinations of dimensions is desirable, but can be problematic. From this discussion we identify the fundamental problem underlying real-time AI research: while real-time guarantees involve resource bounds, AI methods emphasize solution quality.

In Section 5 we show how this problem makes simple tactical methods incapable of meeting the demands of real-time AI systems. We then discuss ways of combining strategic and tactical methods to provide the type of flexible performance guarantees required by real-time AI. We consider the existing methods of deliberation scheduling [2] and imprecise computation [7], and use their features to motivate our more general approach to real-time AI, embodied in the Cooperative Intelligent Real-Time Control Architecture (CIRCA) [11].

Throughout this paper we will provide examples drawn from a domain in which a Hero 2000 robot undertakes delivery tasks, trying to carry objects between rooms in an office building. The robot navigates through hallways using an aimed sonar sensor, trying to avoid collisions with obstacles and walls. A prototype implementation of the architecture described in Section 6 has controlled both a simulated and a real Hero robot performing these navigation tasks.

2 Any-Resource Algorithms

On-line performance guarantees are often defined by the conditions that determine when a system's control algorithm returns a result. We have noted, for example, that an any-time algorithm guarantees that it will use only a bounded amount of time by performing iterative computations that can return a result whenever an external mechanism interrupts the algorithm. For example, in the hall-following domain, an any-time algorithm might be used to check arriving sonar data for obstacles and incrementally update a measure of confidence in the safety of the robot's path. An asynchronous speed-control process would periodically interrupt the any-time algorithm to request the current confidence measure and decide whether the robot should halt.

Of course, time is not the only resource that may be limited for a system: other bounded resources might include memory and non-computational physical features like sensors and actuators. We can generalize the any-time concept to provide guarantees on other resource usage dimensions by noting that any-time algorithms have two crucial elements: an iterative computation producing intermediate results, and an asynchronous mechanism monitoring the time and interrupting the iteration when the deadline is reached. So an *any-resource* algorithm is composed similarly of an iterative computation and an asynchronous monitor that may keep track of any resource measure, and will interrupt the iterative computation when some resource threshold is reached. Any-resource algorithms can thus guarantee that they will not exceed some maximum level of resource usage. Figure 1 illustrates a generic any-resource algorithm in pseudo-code.

As another example of an any-resource method, consider a scenario in the hall-following domain where the robot's delivery tasks have no pressing deadline, but the control system has a limited memory. An any-memory algorithm would be useful

```
on interrupt return(best-partial-result);
while (T)
 {
   new-partial-result = compute-next-result-from(best-partial-result);
   best-partial-result = best-of(best-partial-result, new-partial-result)
 }
```

In parallel with:

```
if (resources-used >= max-resource-threshold) then interrupt;
```

Figure 1: Pseudo-code for the parallel elements of a generic any-resource algorithm.

for planning paths in this situation, because the planning algorithm could require exponential amounts of memory as it constructs and stores alternative partial paths. If the path planner writes beyond the free memory, it might corrupt critical control data and cause a catastrophic failure. The any-memory planning algorithm would use an external (operating system) monitor to keep track of the available memory and, when it ran low, the monitor would interrupt the iterative path planner and use the most recent partial plan. Thus the any-memory algorithm guarantees that the system will not exceed the available memory capacity.

Because the resource monitoring for any-resource algorithms is generally done by an asynchronous process outside of the iterative loop, these methods have the advantage that they do not depend on *a priori* knowledge of the resources available. Unlike strategic scheduling methods, which must be given information about the total available resources and resource requirements, any-resource algorithms can make performance guarantees even when resource limits are dynamic or are unknown when the algorithm starts. Any-resource algorithms dynamically adjust their resource usage to avoid exceeding some maximum level determined outside of the algorithm. Thus any-resource algorithms are particularly appropriate for tasks where multiple computations may be competing for resources; the any-resource algorithms will automatically avoid over-taxing resources.

3 Any-Quality Algorithms

Unfortunately, any-resource algorithms do not provide any control over their output quality; whenever an any-resource algorithm's resource threshold is reached, it returns the current result, which may have less-than-optimal precision, confidence, completeness, or other quality measures. If result quality is critical, any-resource algorithms are inappropriate.

```
while (quality(best-partial-result) < min-quality-threshold)
 {
   new-partial-result = compute-next-result-from(best-partial-result);
   best-partial-result = best-of(best-partial-result, new-partial-result)
 }
return(best-partial-result);
```

Figure 2: Pseudo-code for a generic any-quality algorithm.

Any-quality algorithms *can* make output quality guarantees. While similar to any-resource algorithms in that they iteratively compute intermediate results, any-quality algorithms differ in that their termination decisions are made internally, within the iterative loop, rather than externally as in an any-resource algorithm. For any-quality algorithms, the termination condition is specified as a desired minimum level of result quality, rather than a maximum level of resource availability. The iterative loop produces partial results and tests those results against the quality threshold, returning the first partial result that exceeds the specified quality. Figure 2 illustrates a generic any-quality algorithm in pseudo-code.

As a specific example, many iterative numerical methods [1] are any-precision algorithms. An iterative numerical method continually refines its estimate for the solution to a problem until the precision of its estimate is known to be beyond a certain level. In our hall-following domain, the robot might use such an iterative method to compare sonar readings against an internal map to compute its absolute position to within n centimeters. The algorithm would continue running until it achieved that level of accuracy, as opposed to the any-time methods discussed above, which terminate when a deadline is reached. If absolute positioning is critical to the robot's task, then an any-precision algorithm would be appropriate, while if the task has hard deadlines, an any-time algorithm might be better.

Internal, synchronous monitoring is most appropriate for any-quality algorithms because the algorithm only needs to check its threshold condition when a new incremental result has been computed. A continuous asynchronous monitor might check the condition too frequently, wasting effort by repeatedly examining the quality of a result that has not changed. Or, an asynchronous monitor might not run frequently enough, so that some intermediate results would never be checked against the termination threshold. Furthermore, the most logical place to locate the knowledge of a desired result attribute is in the program generating the results, as opposed to some external arbiter.

Just as any-resource algorithms cannot guarantee output quality, a fundamental weakness of any-quality algorithms is that they cannot guarantee limited resource

usage. By definition, any-quality algorithms must consume resources until they achieve the desired quality threshold.

4 Any-Dimension Algorithms and Their Combinations

We use the term *any-dimension algorithms* to refer to the overall class of iterative algorithms that make guarantees along some dimension of performance, either resource usage or output quality. We have noted that a simple any-dimension algorithm has the disadvantage of being unable to control its performance along more than the single dimension specified in its termination conditions.

One approach to fixing this weakness is to combine multiple termination conditions using conjunction and disjunction to yield more interesting algorithmic behavior. Disjunctive (OR) combinations of any-dimension methods lead to a guarantee that crossing one threshold or the other will yield a result. For example, in the hall-following domain, combining any-time and any-confidence conditions might be the most appropriate method for planning parts of a path under time pressure; the resulting algorithm would work on each planning subproblem until it either found a result in which it had sufficient confidence, or until the time allotted to that subproblem expired.

Disjunctive combinations of thresholds are actually quite common. A simple any-quality algorithm will run until its result reaches the quality threshold; if the threshold is too high, the any-quality algorithm may never terminate. Thus, most implementations of any-quality algorithms also include an alternative, resource-based termination condition, so that they will terminate even if their original quality threshold is never reached. For example, an any-precision algorithm might also have a condition that will terminate the algorithm after a certain number of iterations, regardless of the precision that has been reached at that time.

Similarly, a simple any-resource algorithm will run until it has consumed the allocated resources, even if the algorithm finds an optimal (highest quality) result before the resources are exhausted. To avoid this waste of resources, most any-resource algorithms also include a termination condition specifying an acceptable quality measure. For example, an any-time search algorithm might have both a deadline and a termination condition specifying the goal of the search. If the goal is reached before the deadline, the algorithm terminates and returns its result even though it could have used more time.

Conjunctive (AND) combinations of any-dimension methods lead to guarantees over multiple dimensions. For example, combining any-confidence and any-precision conditions leads to results with guaranteed precision and confidence: the algorithm will continue until both thresholds are reached. However, if we try to conjoin any-time

and any-precision algorithms, we will not necessarily obtain both guaranteed precision and guaranteed timeliness. What happens if the time threshold (deadline) is reached before the precision threshold? The deadline indicates that all the allocated resource (time) has been consumed. If the algorithm terminates it fails to achieve the desired precision, but if it continues it will violate the resource threshold. This example illustrates a fundamental restriction on conjunctive combinations: they cannot be applied to any-resource algorithms, because resource thresholds represent maxima. However, as described above, conjunctions can be applied to any-quality methods because the thresholds on those dimensions are minima, and exceeding minimal quality is generally desirable.

5 Any-Dimension Algorithms and Real-Time AI

The inability to build conjunctions of any-resource and any-quality algorithms is at the heart of why real-time AI is so elusive. Real-time systems require resource-usage guarantees; they must produce a result "by the right time." AI, on the other hand, is concerned with solution quality: a chess program should make *good* moves, an autonomous vehicle should turn in the *correct* direction to avoid a collision, etc. So AI systems are designed to "do the right thing[1]." Together, real-time AI systems must "do the right thing, by the right time."

But we have shown that, with tactical any-dimension algorithms, guarantees on resource usage and output quality cannot simply be conjoined. The only way around this problem is to remove one of the dimension termination conditions. One approach to doing this is to map one dimension onto another to reduce the conjunctive any-dimension algorithm to testing a single dimension. That is, if we can convert a termination condition expressed in a quality dimension into an equivalent *minimum* level of resource usage, then we know that reaching the minimum resource threshold will ensure also passing the minimum quality threshold. Figure 3 illustrates this mapping operation. Note, however, that now we not only have our usual maximum resource threshold for the any-resource algorithm, but we also have a minimum resource threshold to capture the any-quality dimension. In Figure 3, the algorithm must be restricted to the shaded area. To meet the quality requirement, we have to guarantee that the algorithm will not be interrupted before it reaches the minimum resource threshold. But, since the resource monitor that interrupts the computation is acting asynchronously, how can we make such a guarantee?

To answer this question, recall that in Section 1 we distinguished between strategic and tactical methods for resource-bounded computation. We focused on tactical (any-dimension) approaches because strategic approaches make inflexible assumptions ill-suited to complex, unpredictable control tasks, and because they can lead to severe un-

[1]By "the right thing," we mean the best choice given the system's limited knowledge and resources.

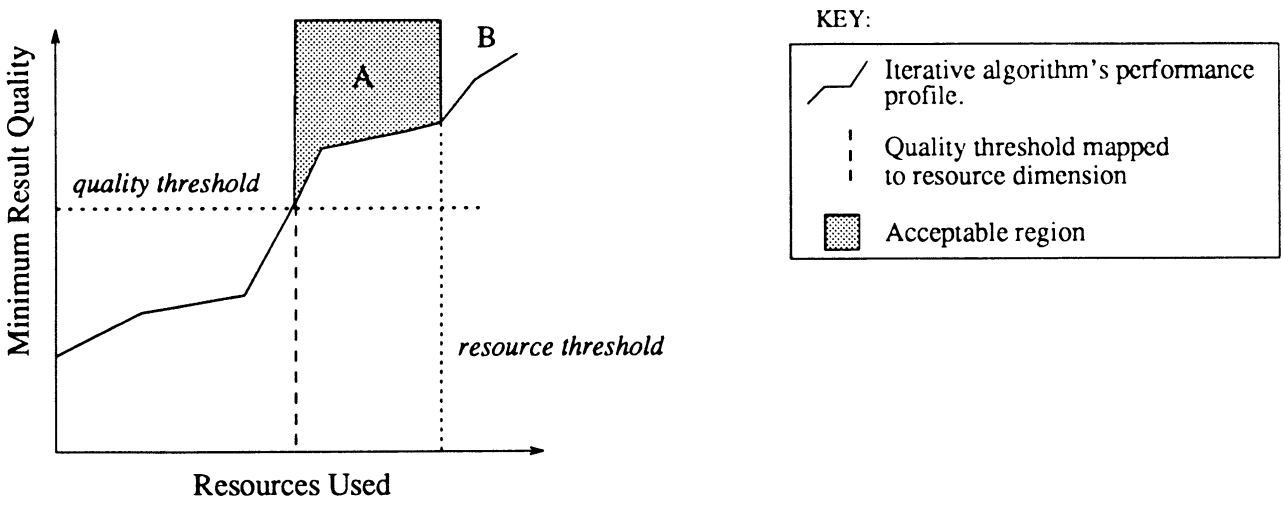

Figure 3: An example performance profile, showing how a quality threshold can be mapped to a minimum resource threshold.

derutilization of resources when high-variance AI methods are employed [12]. However, strategic approaches *can* simultaneously guarantee result quality and resource usage. The next logical step, then, is to try to combine the advantageous features of both strategic and tactical methods to yield a new, combined approach that successfully addresses the requirements of real-time intelligent control in dynamic domains.

Given an any-resource algorithm with both minimum and maximum resource requirements, one approach is to use a strategic method to schedule enough of the resource to assure the minimum threshold, and then to employ a tactical method beyond that to dynamically take advantage of additional resources at runtime. This is essentially the approach taken by Liu *et al.* [7] in the "imprecise computation" method. In this paradigm, an algorithm is divided into mandatory computations that are required to reach a minimal quality threshold, and optional computations that incrementally improve the result and can be interrupted at any time. The imprecise computation scheduler builds schedules that allocate at least enough time for all the mandatory computations. Excess time is scheduled for optional computations.

While the imprecise computation approach has the advantage of balancing strategic and tactical considerations to assure minimum quality within resource bounds, it does not provide any method for dealing with the problems that arise when resources are so scarce that all mandatory computations cannot be scheduled. In this over-constrained situation, an intelligent system must make tradeoffs between the level of output quality it will guarantee and the resource usage it schedules. For example, the system might use load-shedding methods [3, 5, 8] to drop or postpone some mandatory task, leaving resources available for the rest. Or, if alternative methods are available for accomplish-

ing a particular task, the system might attempt to schedule lower-cost methods that will produce a lower-quality solution [3, 9]. When making these tradeoffs between solution quality and resource usage, an intelligent system should use principled methods to decide what it will accomplish.

One approach to dealing with over-constrained systems is to make no guarantees of minimum quality, but instead strive to perform "as well as possible" with the given resources. Dean and Boddy's work on "deliberation scheduling" [2] uses decision-theoretic methods to build task schedules that optimize a measure of overall system utility (output quality). The various problem-solving methods that a system might need to run in some situation are cast as any-time algorithms. The deliberation scheduling problem is then to decide how long each competing any-time algorithm should be run. Dean and Boddy assume that a performance profile, like the one in Figure 3, is available for each system task, and that these tasks are interruptible, restartable, and completely independent, so that the total system utility is simply the sum of the utility levels achieved by individual tasks. Given these assumptions, a scheduling algorithm can maximize system utility by running, at each moment, the task with the largest expected gain in utility. In over-constrained systems, the any-time algorithms will continue to guarantee output timeliness, but output quality will be sacrificed as much as necessary to meet the deadline.

Thus, while imprecise computation assures minimum solution quality given minimal resources, deliberation scheduling commits to doing as well as it can given no assumptions on resources. Both approaches assume that the system is given a fixed mapping between the output quality (utility) dimension and the resource usage (time) dimension. There are two fundamental problems with this assumption. First, such mappings may be difficult or impossible to derive, because the performance of most algorithms is highly dependent on the particular problem to which the algorithm is being applied. For example, Liu *et al.* [7] describe an any-time implementation of Newton's method for finding the roots of a function F. Unfortunately, as illustrated in Figure 4, the number of iterations this method requires to achieve a result with specified precision is highly dependent on both the domain (the function F) and the internal state of the system (the initial guess for the root value). Because the precision threshold cannot be mapped onto the time dimension, the root-finding computation cannot be cleanly separated into mandatory and optional parts based on time alone.

The second major difficulty is that, even if an individual algorithm's output quality can be accurately characterized by a fixed performance profile, tasks are not independent in realistic domains; the utility of a particular computation depends on other task computations. In our robot domain, the utility of running a computation to decide whether to halt the robot is dependent on whether the routine that locates obstacles has been run. Furthermore, the utility of the obstacle-locating routine is affected by the

```
xguess = initial_xguess;
while (abs(xnew - xguess) > .01)
      {
         xguess = xnew;
         xnew = xguess - F(xguess) / Fprime(xguess);
      }
```

(a) Newton's method.

	initial_xguess		
F	1	10	20
x^2	7	10	11
$e^x - 1$	4	13	23
$e^{25x} - 1$	27	252	502

(b) Iterations to achieve .01 prec

Figure 4: Showing the difficulty of mapping precision to time for Newton's root-finding method.

fact that its results will be used to decide about halting the robot. These routines have high utility when used in conjunction, in a particular order, but low utility otherwise.

Our approach to real-time AI, embodied in CIRCA, combines features of deliberation scheduling and imprecise computation, but avoids assumptions about task independence and performance profiles. CIRCA allows arbitrarily complex decision-making about tradeoffs between possible solutions given particular resource bounds.

6 CIRCA

In this section, we provide a high-level description of CIRCA and show how the architecture combines both strategic and tactical resource-management methods to implement real-time control using complex AI methods. To clarify this discussion, we include details from our prototype implementation of CIRCA that controls a Hero robot performing the hall-following task. In this domain, the system must actively acquire data from its sensors, process that input data, and produce responses to drive its actuators and affect the external world. These control behaviors are divided into a set of "reactive behaviors," or reactions, that look for certain inputs and produce appropriate responses. Some of these reactions will produce outputs that must meet domain-defined hard deadlines; for example, the reactions that recognize obstacles and halt the robot must execute frequently enough to avoid collisions.

More details on CIRCA are available in [11][2], and [10] presents a complete description of the flexible world modeling methods the system uses to make heuristic decisions about trading off resource usage and output quality.

Figure 5 illustrates the architecture, in which an AI subsystem (AIS) and Scheduler

[2]This journal article is still in press. A compressed, postscript version of the manuscript is available via anonymous ftp to ftp.eecs.umich.edu in the file outgoing/djm/circa.ps.Z .

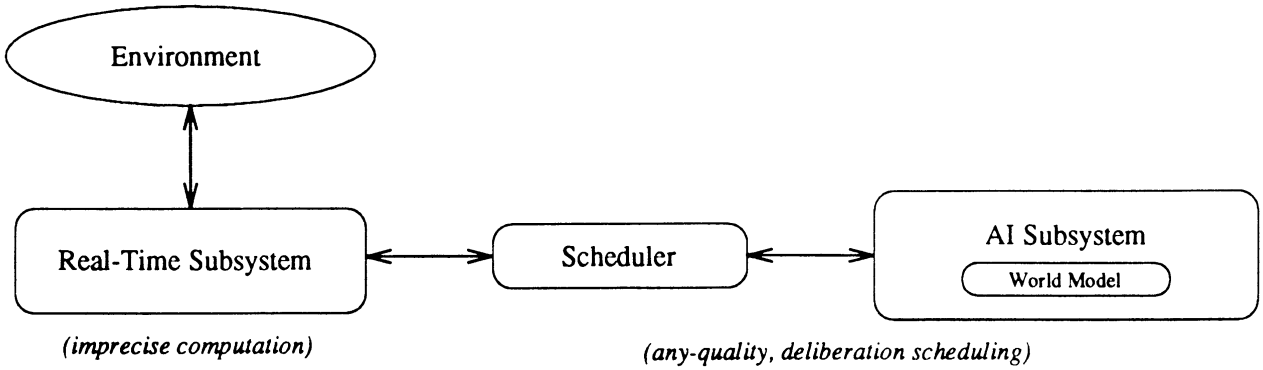

Figure 5: Overview of CIRCA.

cooperate to strategically select and schedule a set of tasks that will cope with a particular expected domain situation. The parallel real-time subsystem (RTS) is responsible for precisely executing the task schedules, ensuring that a schedule that theoretically meets the demands of the domain will operate correctly in the actual domain. In the following subsections, we show how the AIS and Scheduler incorporate an any-quality algorithm, and how they operate as a generalized deliberation scheduling system that permits dynamic dependencies between scheduled tasks. We also show how the RTS implements tactical methods resembling the imprecise computation technique, to take advantage of resources that become available during task execution.

6.1 The AIS/Scheduler and Deliberation Scheduling

Unlike deliberation scheduling, CIRCA does not require task performance profiles, and is capable of building useful guaranteed task schedules even when tasks have very complex interactions and dependencies. Deliberation scheduling is able to analytically derive an optimal schedule given performance profiles; CIRCA requires less precise information and performs a *search* for a desirable task schedule that yields acceptable output quality within given resource bounds.

CIRCA implements this search by iterating over a loop that first has the AIS plan a set of tasks to meet a given output quality threshold, and then runs the Scheduler on those tasks to see if they can all be successfully run given the system's limited resources. In essence, this process corresponds to choosing a point along the performance profile (for the overall system, not a single task) that is above the quality threshold, and then using the Scheduler to check if that point is also below the maximum resource usage threshold (e.g., point A in Figure 3). Failure to produce a schedule is an indication that the chosen set of tasks, while providing sufficient output quality, requires too many resources (e.g., point B in Figure 3). This iterative process of choosing a set of tasks to achieve a given level of output quality and then checking their resource usage with the Scheduler can be viewed as an any-quality algorithm: the iteration will continue until a schedule of tasks is found that exceeds the desired quality threshold. Because

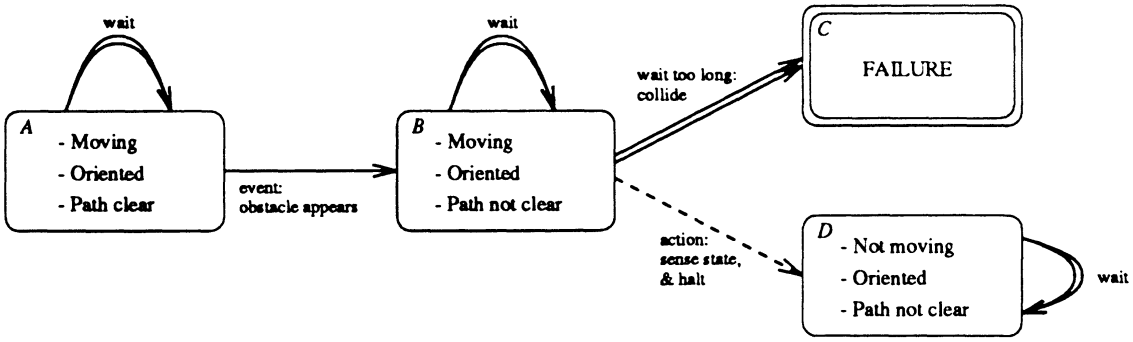

Figure 6: An abstracted portion of the AIS' world model for the example domain. For clarity, many states, state features, and transitions have been omitted.

this generate-and-test technique does not rely on explicit knowledge of the form of the performance profiles for each task, it is more widely applicable than the deliberation scheduling technique.

Rather than performance profiles, the AIS uses a general world model to decide what tasks should be active at any time. This causal model represents both the effects of the system's possible computations and the events that may occur in the world. In the prototype implementation, the world model takes the form of a directed graph in which nodes represent possible states of the world and arcs represent transitions between those states. The model distinguishes three distinct types of state changes: *action transitions*, performed deliberately by system reactions; *event transitions*, due to external world occurrences; and *temporal transitions*, due to the passage of time. Figure 6 represents a small portion of the graph model for the hall-following domain, showing the motivation for the system's obstacle detection routines. In the figure, an event transition (solid arrow) from state A to state B indicates the state change caused when an obstacle appears in the path; if the system waits too long to recognize the situation and take action, it will follow the temporal transition (double-arrow) to failure by colliding with the obstacle. When building this world model to choose which reactions should be active, the AIS recognizes the possibility of failure, and plans a task that will quickly detect the obstacle and perform the `HALT` action transition (dashed arrow) to avoid a collision, moving to state D instead.

The world model thus represents the motivations for tasks by their complex causal interactions with the system's internal state, other tasks, and the domain. In addition to avoiding reliance on independence assumptions and simple performance profiles to motivate task choices, the world model and AIS planning mechanism provide a basis for informed, intelligent decisions about the tradeoffs necessary in over-constrained domains where the available resources are not sufficient to guarantee optimal output

quality. CIRCA generalizes the "best effort" approach of deliberation scheduling, allowing arbitrarily complex reasoning about what levels of output quality are acceptable, and how output quality and resource usage should be traded off in an over-constrained system.

Initially, the AIS will plan a set of tasks that meets a high level of desired quality. If the Scheduler indicates that those tasks require too many resources, the AIS then has several options. If alternative computational tasks are available to address the original goal, the AIS might continue the any-quality iterations described above, generating and testing new sets of tasks that meet the quality threshold. In the hall-following example, this might correspond to planning alternative computational tasks to implement the same world model transitions. If the any-completeness algorithm has not found such a schedule after some time, the AIS also has the option of altering the quality threshold it is seeking to guarantee. For example, the hall-following robot might initially plan to traverse a hallway while watching for obstacles and counting doorways along the way. If the AIS/Scheduler any-completeness iteration process searches for, but cannot find, a schedule of tasks to accomplish these goals, the system may decide that the doorway-counting task is less important than obstacle detection, and choose to ignore the doorway task. In essence, the system makes a dynamic decision to lower the threshold of acceptable output quality. By making this adjustment, the system can build a schedule of tasks that is still guaranteed to achieve the reduced quality goal, and thus at least some of the guaranteed nature of the system is preserved.

Alternatively, the AIS might decide that counting doorways and avoiding obstacles are both crucial to its goals (perhaps because it must turn at the third doorway), so it might slow the robot down to give it more time for those tasks. In this case, the quality of the solution has been lowered because the robot will arrive at its destination later, but it will have the required knowledge about doorways. This example shows how CIRCA deals with complex interactions among tasks: the utilities of the individual tasks that count doorways and watch for obstacles are highly dependent on each other and on the larger goal-directed context in which they are used. By representing the causal interactions of the system's tasks and the domain, CIRCA's world model provides a principled basis for making quality/resource tradeoffs.

6.2 The RTS and Imprecise Computations

Once an acceptable task schedule has been produced by the AIS and the Scheduler, the RTS is responsible for executing the schedule. However, we have previously noted that just using strategic scheduling methods can lead to low resource utilization. For example, if no obstacles appear in the robot's path, the collision-avoidance reaction planned for the example scenario will check for obstacles frequently but its `HALT` action will never need to be executed, and the execution time and other resources reserved in

the schedule for that action will never be used.

To take advantage of these unused resources, the RTS implements a generalized imprecise computation technique. In addition to the schedule of tasks that must be guaranteed to meet their deadlines, the AIS can also send the RTS a list of "if-time" or "best-effort" tasks, that should be run only if unused resources become available. When resource limits force the AIS to reduce the expected output quality of the tasks it has planned, it adds to the if-time list those tasks which could improve the output quality. When the RTS is executing the schedule of mandatory tasks, it checks after completing each task to see if the task has used less than its assigned time[3]. If so, the RTS will use that time to find and execute if-time tasks that will fit into the remaining, unused time.

In the hall-following example where the system cannot guarantee to count doorways, the AIS will add the doorway-counting task to the if-time list. Then, while the schedule of guaranteed tasks is being executed on the RTS, if a particular execution of the obstacle detection task does not use all of its scheduled time, the RTS will check to see if the remaining time (and other resources) are enough for the doorway-counting task. If so, that if-time task will be executed, and the system will have accomplished, at run time, a higher level of output quality and resource utilization than could be guaranteed by the strategic scheduler. Of course, in a demanding domain with many obstacles, the doorway-counting task might never be executed; as a best-effort task, it serves only to improve output quality, and it is not guaranteed to run.

Thus the RTS relies on a tactical method to increase the utilization of resources that become dynamically available at runtime. Viewed from a high level, the RTS functions as a generalized form of imprecise computation: the guaranteed tasks represent mandatory computation, and the if-time tasks are optional computations. However, while the imprecise computation method specifies that optional computations are any-time algorithms that will improve the quality of the mandatory computations they follow, CIRCA's if-time tasks need not be incremental, and they may have little or no relation to the tasks they follow. If-time tasks are simply those tasks which the AIS/Scheduler decided were desirable, but which could not be fit into the schedule of guaranteed tasks.

7 Conclusion

By describing the class of any-dimension algorithms, we have focused attention on the performance guarantees made by these tactical resource management methods. In examining the weaknesses of any-dimension algorithms, we have found that conjunctive combinations of resource guarantees are not possible with tactical methods. Thus we recognize that the fundamental problem underlying attempts to develop real-time AI

[3]This will often be the case, because mandatory tasks are scheduled based on their worst-case execution time, which may be much larger than the average execution time [4].

systems is the need for both guaranteed maximum resource usage and guaranteed minimum output quality.

Because simple tactical methods cannot provide this conjunction of guarantees, and simple strategic methods are too inflexible for dynamic, unpredictable domains, we propose that the best approach to real-time AI is to combine the features of these methods. The imprecise computation method provides such a combination, but has no way to handle over-constrained systems where a given level of quality may not be possible. The deliberation scheduling method does handle over-constrained systems, but to do so it requires task independence and static performance profiles for the tasks it is scheduling. These requirements may not be met by realistic domains.

CIRCA combines strategic and tactical methods, provides principled methods for handling over-constrained domains, and does not require performance profiles. To realize these functions, CIRCA relies on a causal world model to represent complex task dependencies and utilities. Using this world model to drive planning and scheduling, CIRCA searches for task schedules that maximize guaranteed output quality given limited resources. While executing those schedules of guaranteed tasks, CIRCA also provides tactical methods that can take advantage of reserved but unused resources to further improve output quality.

We have implemented a prototype of CIRCA, and have shown that it successfully navigates and avoids collisions in the hall-following domain described above. We are currently improving the task planning and world modeling mechanisms to address complexity issues. In addition, the RTS is being ported to a VxWorks-based real-time testbed, which will allow us to verify the system's timing guarantees.

References

[1] R. L. Burden and J. D. Faires, *Numerical Analysis*, PWS-KENT Publishing Co., 1989.

[2] T. Dean and M. Boddy, "An Analysis of Time-Dependent Planning," in *Proc. National Conf. on Artificial Intelligence*, pp. 49–54, 1988.

[3] A. Garvey and V. Lesser, "Design-to-time Real-Time Scheduling," *to appear in* IEEE Trans. Systems, Man, and Cybernetics, 1992.

[4] D. Haban and K. G. Shin, "Application of Real-Time Monitoring to Scheduling Tasks with Random Execution Times," *IEEE Trans. Software Engineering*, vol. 16, no. 12, pp. 1374–1389, December 1990.

[5] T.-W. Kuo and A. K. Mok, "Load Adjustment in Adaptive Real-Time Systems," in *Proc. Real-Time Systems Symposium*, pp. 160–170, December 1991.

[6] K.-J. Lin, S. Natarajan, and J. W.-S. Liu, "Imprecise Results: Utilizing Partial Computations in Real-Time Systems," in *Proc. Real-Time Systems Symposium*, pp. 210–217, December 1987.

[7] J. W.-S. Liu, K.-J. Lin, and S. Natarajan, "Scheduling Real-Time, Periodic Jobs Using Imprecise Results," in *Proc. Real-Time Systems Symposium*, pp. 252–260, December 1987.

[8] J. W. S. Liu, K.-J. Lin, W.-K. Shih, *et al.*, "Algorithms for Scheduling Imprecise Computations," *IEEE Computer*, vol. 24, no. 5, pp. 58–68, May 1991.

[9] N. Malcolm and W. Zhao, "Version Selection Schemes for Hard Real-Time Communications," in *Proc. Real-Time Systems Symposium*, pp. 12–21, December 1991.

[10] D. J. Musliner, E. H. Durfee, and K. G. Shin, "World Modeling for the Dynamic Construction of Real-Time Control Plans," submitted to *Artificial Intelligence*, 1992.

[11] D. J. Musliner, E. H. Durfee, and K. G. Shin, "CIRCA: A Cooperative Intelligent Real-Time Control Architecture," *IEEE Trans. Systems, Man, and Cybernetics*, 1992 (in press).

[12] C. J. Paul, A. Acharya, B. Black, and J. K. Strosnider, "Reducing Problem-Solving Variance to Improve Predictability," *Communications of the ACM*, vol. 34, no. 8, pp. 81–93, August 1991.

[13] S. J. Russell and S. Zilberstein, "Composing Real-Time Systems," in *Proc. Int'l Joint Conf. on Artificial Intelligence*, pp. 212–217, August 1991.

[14] J. A. Stankovic, "Misconceptions about Real-Time Computing: A Serious Problem for Next-Generation Systems," *IEEE Computer*, vol. 21, no. 10, pp. 10–19, October 1988.