Managing Online Self-Adaptation in Real-Time Environments

Robert P. Goldman, David J. Musliner, Kurt D. Krebsbach
Automated Reasoning Group
Honeywell Technology Center
3660 Technology Drive
Minneapolis, MN 55418
{goldman,musliner,krebsbac}@htc.honeywell.com

1 Introduction

Suppose we have an autonomous aircraft flying a complex mission broken into several different phases such as takeoff, ingress, target surveillance, egress, and landing. For each mission phase, the aircraft's control system will have prepared a plan (or controller) specifying particular actions and reactions during the phase. Now suppose that the autonomous control system onboard this aircraft is self-adaptive: that is, it can modify its own behavior (plans) to improve its performance. Why might it have to adapt? Perhaps because the mission is changed in-flight, perhaps because some aircraft equipment fails or is damaged, perhaps because the weather does not cooperate, or perhaps because its original mission plans were formed quickly and were never optimized. In any case, the aircraft's self-adaptive control system is now facing a *deliberation scheduling* problem. It must decide which mission phase's plan to try to improve via self-adaptation, how to improve that plan, and how much time to spend on that self-adaptation process itself.

This deliberation scheduling problem has two strong real-time components. First, the deliberation scheduling process must take into account the time that self-adaptation will require: the value of the adaptation is affected by the time at which it can be produced, and its relationship to alternative uses for that computation time. Second, the deliberation scheduling process itself (i.e., deciding what to think about) consumes time, and hence affects the potential value of self-adaptation.

We are developing the Self-Adaptive Cooperative Intelligent Real-Time Control Architecture (SA-CIRCA) to address precisely this type of domain, including the deliberation scheduling problem [7, 8]. SA-CIRCA is a domain-independent architecture for intelligent, self-adaptive autonomous control systems that can be applied to hard real-time, mission-critical applications. SA-CIRCA includes a Controller Synthesis Module (CSM) that can automatically synthesize reactive controllers for environments that include timed discrete dynamics. This controller synthesis process can occur both offline, before the system begins operating in the environment, and online, during execution of phase plans. Online controller synthesis is used to adapt to changing circumstances and to continually improve the quality of controllers for current and future mission phases.

SA-CIRCA's Adaptive Mission Planner (AMP) is responsible for the highest-level control of an SA-CIRCA agent, decomposing the overall mission into phases while managing the agent's responsibilities (by negotiating with other agents) and its deliberation activity. We are currently developing the AMP's deliberation scheduling functions, emphasizing the real-time aspects of the problem. An experimental SA-CIRCA module uses stochastic models of the controller synthesis process to allocate computational effort to controller improvement across the mission phases. The synthesis process model addresses the first real-time aspect of deliberation scheduling: it attempts

to predict how long the controller synthesis process will take for a particular type of improvement for a particular mission phase. This information can then be used in a decision-theoretic estimate of the expected utility for a proposed self-adaptation.

In this paper we address the second real-time aspect of deliberation scheduling, i.e., the time cost of the meta-level decision itself, by developing computationally feasible heuristics that make deliberation scheduling decisions "greedily" but quickly. To assess the performance of these greedy heuristics, we are building somewhat simplified Markov Decision Process (MDP) models of the AMP's deliberation scheduling problem and assessing both the optimal and greedy solution policies. Our preliminary results indicate that these greedy heuristics are able to make high-quality deliberation scheduling decisions in polynomial time, with expected utility measures quite close to the NP-complete optimal solutions.

The SA-CIRCA agent was designed to control mission-critical systems, under time-pressure. In our current experiments, we are working with teams of SA-CIRCA agents controlling simulated Uninhabited Aerial Vehicles (UAVs), on combat missions. SA-CIRCA automatically and dynamically generates hard real-time control programs that are guaranteed to keep the platform safe while it executes the mission. The SA-CIRCA State-Space Planner (SSP) generates these controllers from models of the system's environment, and their guarantees are provided based on control and timing information in those models. For example, the SA-CIRCA UAV has a model of radar-guided SAM threats, containing information about how fast these threats operate (see Figure 1). Among other things, the model specifies the minimum delay between the UAV's detecting that it has been "painted" by enemy radar, and its destruction by a missile of this type. SA-CIRCA uses this information to ensure that its controllers monitor enemy radar lock-ons frequently enough, and take countermeasures fast enough.

SA-CIRCA does not have unlimited resources at its disposal. In particular, it suffers from the problem of *bounded reactivity* [6]: it can only monitor and react to a limited number of threats concurrently. To overcome this problem, SA-CIRCA's Adaptive Mission Planner decomposes extended missions into multiple mission phases, each of which can have a different controller tailored to its particular needs. See Figure 2. For example, an air mission with high and low altitude components would have different controllers for each. In the high altitude phase the UAV would monitor radar-guided SAM threats, whereas at low altitude these would not be relevant. At low altitudes, on the other hand, the aircraft would be relatively safe from radar-guided SAMs, but would have to guard against shoulder-launched, IR-guided SAMs.

In order to deal with dynamic situations about which we have only limited information, SA-CIRCA is able to tailor its mission phase controllers on line. For example, while in the process of traversing enemy airspace on the way to a target, the agent may be informed of a previously unknown SAM site on its exit path. The agent will generate a new controller for the egress phase of its mission that will be able to handle this threat.

Due to bounded resources, the opportunity of dynamic adaptation poses the corresponding problem of *deliberation scheduling* [1, 4]. The SA-CIRCA agent must determine how best to allocate its limited computational resources to improving controllers for various mission phases. For example, should the agent first improve the controller for the final phase, since it is perceived to be

```
;; to evade simple radar missiles, start the reliable temporal transition
;; and then, after it is done, resume normal path.
(make-instance 'action
            :name "begin_evasive"
            :preconds '((path normal))
            :postconds '((path evasive))
            :delay 10)

(make-instance 'reliable-temporal
            :name "evade_radar_missile"
            :preconds '((radar_missile_tracking T) (path evasive))
            :postconds '((radar_missile_tracking F))
            :delay (make-range 250 400))

(make-instance 'action
            :name "end_evasive"
            :preconds '((path evasive))
            :postconds '((path normal))
            :delay 10)

;; The radar threats can occur at any time.
(make-instance 'event
            :name "radar_threat"
            :preconds '((radar_missile_tracking F))
            :postconds '((radar_missile_tracking T)))

;; If you don't defeat a threat by N seconds, you're dead.
(make-instance 'temporal
            :name "radar_threat_kills_you"
            :preconds '((radar_missile_tracking T))
            :postconds '((failure T))
            :delay 1200)
```

Figure 1: Example of the CIRCA description of a radar-launched SAM threat.

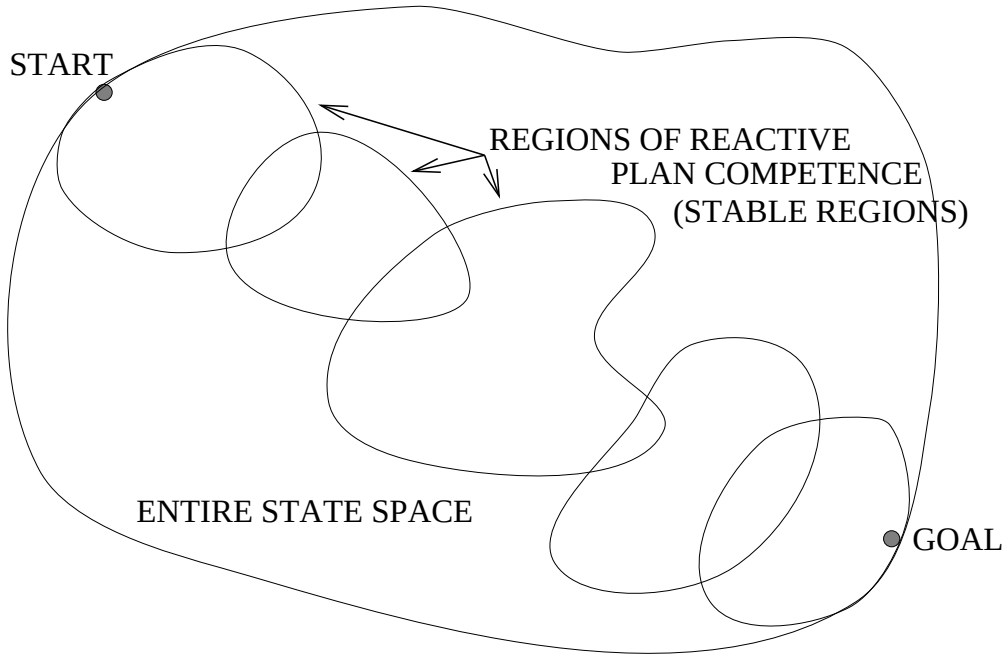

Figure 2: Decomposition of a mission control problem into multiple single-phase control problems.

the worst, or should it allocate those resources to polishing the controller for an earlier phase, knowing that it can work on the final phase later? This paper describes our initial work on deliberation scheduling for the SA-CIRCA architecture.

2 The Adaptive Mission Planner (AMP)

The CIRCA Adaptive Mission Planner (AMP) is responsible for the highest-level control of a CIRCA agent [7, 8], determing and modifying the agent's responsibilities (threats to handle, mission goals to achieve), controlling the agent's reasoning (what plans to construct), and managing the agent's deliberation resources (i.e., how best to use computation time to improve the overall mission plan). More specifically, the AMP manages the agent's responsibilities by negotiating with other agents via contract bidding. It controls the agent's reasoning both by modifying problem configurations for the CSM, and by invoking (or halting) the CSM when appropriate. Finally, the AMP manages the agent's deliberation resources by scheduling the CSM to improve certain plans in a manner that yields the highest utility for the mission plan as a whole.

We are working on bringing the planning activity (deliberation) of the Controller Synthesis Module (CSM) under real-time control, so that the Adaptive Mission Planner (AMP) can manage and control that deliberation via a negotiation process. The AMP decomposes the problem into appropriate phases, for which the CSM generates safety-preserving real-time control plans. This planning process is performed prior to execution, and continues as the system is executing other portions of the high-level plan. Together, the AMP and CSM cooperate to effectively allocate this planning effort across the entire high-level plan while meeting intermediate deadlines imposed by execution-time constraints.

2.1 Problem Structure

The overall team mission is divided into *phases*, which correspond to modes or time intervals that share a fundamental set of common goals, threats, and dynamics. For example, our military Unmanned Aerial Vehicle (UAV) scenarios include missions that have phases such as ingress, attack, and egress. The ingress phase is distinguished from the attack phase both by the characteristics of the flight path (e.g., a nap-of-earth stealthy approach vs. a popup maneuver very near a target) and by the expected threats (e.g., the types of missile threats present at different altitudes) and goals (e.g., reaching the target zone vs. deploying a weapon).

2.2 Agent Responsibilities

The team of CIRCA agents must arrange to have different agents take responsibility for different goals and threats, depending on their available capabilities and resources (e.g., ECM equipment and weapons loadout). These goals and threats can vary from one phase to the next, and in fact, the mission is typically split into phases specifically to decompose the overall mission into manageable chunks aligned with a common set of threats, or a common goal which, when achieved, signals the end of that phase.

For each mission phase, the CIRCA agents must have plans, or controllers, that are custom-designed (either before or during mission execution) to execute the mission phase and make the best possible effort to achieve the goals and defeat the threats associated with the phase. The CSM, described elsewhere, is capable of automatically building these controllers, but this controller synthesis can be a complex and time-consuming process. The complexity (and hence duration) of the CSM process can be controlled by varying the *problem configuration* that is passed to the CSM to describe the characteristics of the desired controller for a particular mission phase [5]. The problem configuration contains information about the initial state of the world, goals to achieve, threats that are present, state transitions that can happen due to the world, and actions available to the agent to affect the world. By varying these details, the AMP can make a planning problem fall anywhere in a complexity spectrum from very simple to infeasible.

2.3 Predictive Deliberation Management

Given this, one of the primary responsibilities of the AMP is to determine which mission phase the CSM is trying to build a controller for at any moment, and how hard it should work to do so, by modifying the phase problem configurations. This is what we mean by the AMP's deliberation management function. Our current experiments center around allocating effort to maintain system safety by altering which of the phase's potential threats should be considered, and for how long. In each phase, the more threats that can be handled by improving the current plan for the phase, the better the agent's or the team's survival probability for that phase and for the mission. Thus, the problem can be cast as follows: Given a set of planning phases, quality measures of the current plan for each phase, a set of tradeoff methods (i.e., improvement operators) applicable in each phase, and some amount of time to try to improve one or more of the current plans, how should the AMP allocate the next segment of time to improve the overall expected utility of the mission plan as a whole?

To effectively decide what deliberation should happen now, the AMP must consider the potential

deliberation schedule into the future. For example, the AMP might consider starting a lower-priority CSM task earlier if there is a shorter window of opportunity in which to execute that task, or the expected execution time is longer. In this case, the AMP would also need to consider whether it expects that this will still leave time to execute the higher-priority task later. As we will see, more efficient, but incomplete approaches to the problem can suffer from local maxima, and miss solutions that require this type of further lookahead and more complex analysis.

The second part of the problem is to select which of several improvement operators to apply to the phase it has selected. We assume that each improvement operator (deliberation action) takes an equal amount of time (one quanta), and that the same action can be taken in multiple quanta, if applicable to the phase. In general, action selection is only an interesting decision if there are tradeoffs between the various operators. In our current work, there is a tradeoff between the amount of expected improvement (if successful), and the action's probability of success.

3 A Simple MDP Model of the AMP

Our current experiments on deliberation scheduling are based on a Markov Decision Process (MDP) model of the deliberation scheduling problem. We have posed the problem as one of choosing, at any given time, what phase of the mission plan should be the focus of computation, and what plan improvement method should be used. This decision is made based on where the agent is in its mission and a probabilistic measure of the quality of the plan. In our current model, the quality of the plan is measured by how much reward it is expected to achieve, but the distribution of potential rewards among mission phases is fixed. The system improves its expected reward by reducing its likelihood of failure (which eliminates the possibility of future reward attainment).

In the most abstract form, we formulate the CIRCA deliberation scheduling problem as follows: The AMP has decomposed the overall mission into a sequence of **phases**:

$$\mathcal{B} = b_1, b_2, ..., b_n$$

The CSM, under the direction of the AMP, has determined an initial **plan**, P^0 made up of individual state space plans, p_i^0, for each phase $b_i \in \mathcal{B}$:

$$P^0 = p_1^0, p_2^0, ..., p_n^0$$

P^0 is an element of the set of possible plans, $\mathcal{P}$. In general, it may not be possible to enumerate this set, and it certainly will not be possible to efficiently represent the set. However, we will see that we can usefully reason about classes of plans that are equivalent with respect to a measure of their quality. We refer to a $P^i \in \mathcal{P}$ as the overall **mission plan**. The state of the system, then, may be represented as an ordered triple of time index, the phase in which the agent is currently operating, and the current mission plan:

$$\mathcal{S} = \langle t, b_i, P^t \rangle, t \in \mathbb{N}, b_i \in \mathcal{B}, P^t \in \mathcal{P}$$

With some abuse of notation, we refer to the components of a state using operators $t(S), b(S)$ and $P(S)$ for $S \in \mathcal{S}$.

For simplicity's sake, we assume that the duration of each of the mission phases is known, and that the mission phases always occur in sequence. I.e., for each phase, b_i, there exists a known

start(b_i), end(b_i) and dur(b_i). Therefore, for a given t we can determine the corresponding mission phase (phase(t)) as the b_i satisfying start(b_i) $\leq t \leq$ end(b_i).

The AMP has access to several **plan improvement methods**,

$$\mathcal{M} = m_1, m_2, ..., m_m$$

At any point in time, t, the AMP can choose to apply a method, m_j, to a phase, b_i (written as $m_j(i)$). Application of this method *may* yield a new plan for mission phase b_i, producing a new P^{t+1} as follows: if

$$P^t = p_1^t, p_2^t, \ldots p_i^t, \ldots, p_n^t$$

then

$$P^{t+1} = p_1^t, p_2^t, \ldots p_i^{t+1}, \ldots, p_n^t$$

where

$$p_i^t \neq p_i^{t+1}$$

Note that the application of this method may fail, yielding $P^{t+1} = P^t$. Application of a method never yields a new plan of lower measured quality than the original plan, since if we generate a worse phase plan, we simply discard it and keep the previous one.

Specifically, in CIRCA the available methods are implemented by code which generates a new problem configuration for the AMP to download to the CSM. The CSM will run for a fixed amount of time (a planning "quantum"), and then terminate with a new state space plan, or be interrupted when its quantum has been consumed.

To complete the formulation, we must have a utility (reward) function, U applying to the states. Again, to simplify, we assess the agent a reward, $U(i)$ on the completion of mission phase i. For example, if the aircraft completes its mission successfully and survives, it will receive some reward when completing the final phase. For some missions, it is also appropriate to add some reward to an intermediate phase whose successful completion corresponds to achieving some mission goal (e.g., overflying a reconnaissance target).

Different phases of the mission present different degrees of danger. We represent this by different probabilities of making a transition to a designated failure state. The probability of surviving a given phase is a function of both the hazards posed by that phase, and the quality of the state space plan for that phase. Accordingly, instead of representing the plans explicitly in the system state, S, we need only a vector of survival rates. With some abuse of notation, we will use p_i^t to represent not only the plan itself, but also the associated survival probability, the only aspect of the plan reflected in the MDP. Therefore, the chance of surviving phase i is:

$$\prod_{\text{start}(i) \leq t \leq \text{end}(i)} p_i^t$$

and the expected reward in phase i is:[1]

$$U(i) \prod_{\text{start}(i) \leq t \leq \text{end}(i)} p_i^t$$

[1]Note that this is not the expected *utility*, which must also take into account the chance of the agent being destroyed in this phase.

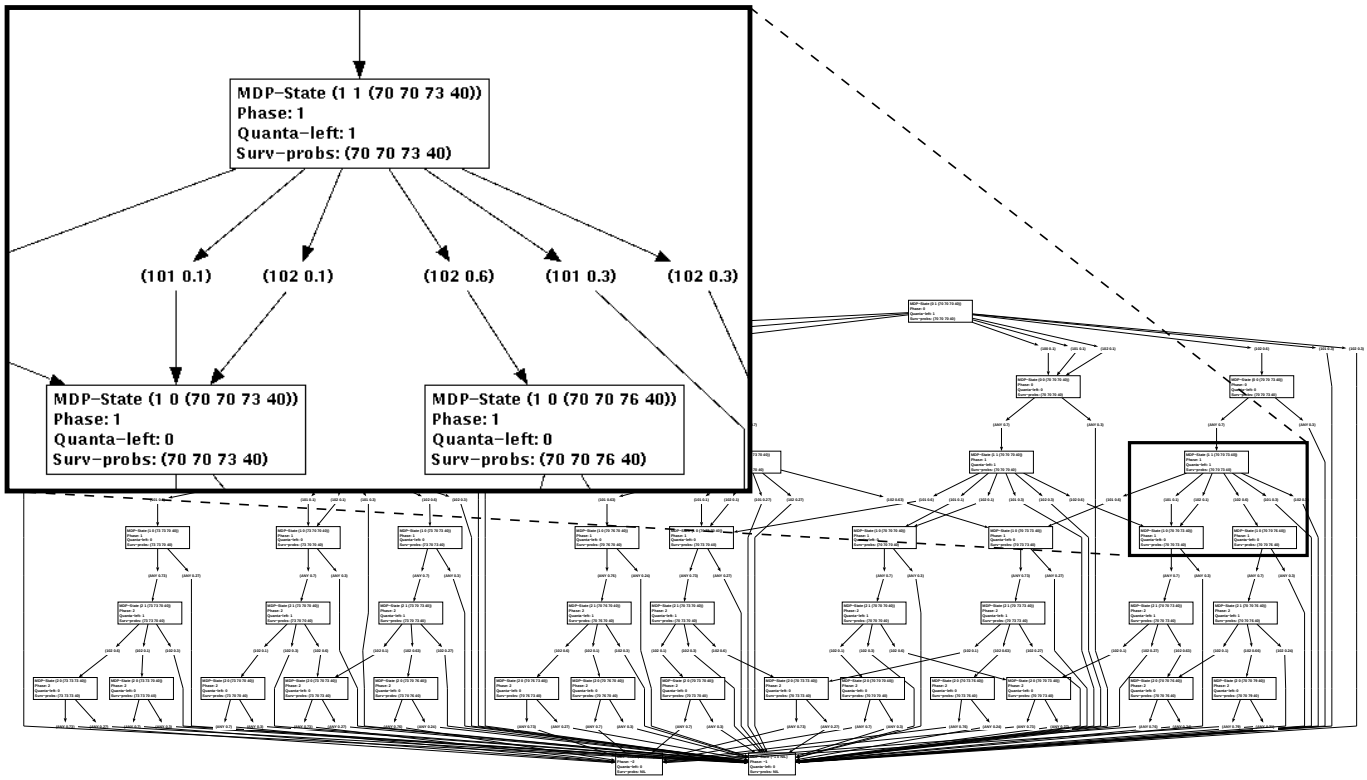

Figure 3: A very simple AMP MDP for 3 phases with one quantum per phase and one deliberation action choice per phase.

Given the representation of plans as survival rates, the plan improvement operators are represented as actions that may take us to a state where the survival probability for a given phase is higher than in the current state. We are not yet considering plan improvement operators that increase the likelihood of achieving reward except by avoiding failure.

Figure 3 illustrates a very simple AMP MDP for 3 phases with one quantum per phase and one deliberation action choice per phase. Even this trivial problem forms a space of 43 states, which makes the diagram unreadable on a page. The zoomed area illustrates how the model works: the two edges labeled `102` leaving the uppermost state `(1 1 (70 70 73 40))` correspond to the AMP deciding to perform deliberation action 1 on phase 2, which is 60% likely to succeed and result in improvement of the phase 2 survival probability (the right edge to state `(1 0 (70 70 76 40))`). Ten percent of the time that action is expected to fail, yielding no survival probability improvement (the left edge to state `(1 0 (70 70 73 40))` and 30% of the time the system is expected to be destroyed during that deliberation quanta (the rightmost edge leading off the zoomed area, to the failure node.

With these simplifying assumptions, the problem of achieving reward is transformed into one of surviving to complete phases that provide reward. The problem of choosing a deliberation action for each state is limited to choosing which phase of the mission plan to try to improve, and which

improvement operator to apply to improve it with. The conventional formulation of an MDP is:

$$\text{policy}^\star(s) = \arg\max_a \sum_{s'} P(s \xrightarrow{a} s')\overline{U}^\star(s')$$

Where a is an action, $P(s \xrightarrow{a} s')$ is the probability of reaching s' from s when the agent executes action a, and $\overline{U}^\star(s')$ is the expected utility of being in state s' while pursuing the optimal policy. With our model of deliberation schedule, we can reformulate the problem as:

$$\text{policy}^\star(s = \langle t, b_k, P^t \rangle) = \arg\max_{i,j}(1 - p^t_{b_k})U(\text{destroyed}) + p^t_{b_k} \sum_{s'} P(s \overset{m_j(i)}{\rightarrow} s')\overline{U}^\star(s')$$

Where i corresponds to the choice of phase to be improved and j the choice of deliberation (planning) operator to use to improve that phase. In our state representation, s' will be constrained to be $\langle t+1, \text{phase}(t+1), P^{t+1} \rangle$ where P^{t+1} is identical to P^t everywhere except for (possibly) phase i. Eliminating constants in the above equation, we have:

$$\text{policy}^\star(s) = \arg\max_{i,j} \sum_{s'} P(s \overset{m_j(i)}{\rightarrow} s')\overline{U}^\star(s')$$

We have conducted several experiments using this formulation of the problem, directly evaluated using value iteration [9]. As one would expect, performance suffers with problem scale (experimental results are reported in the next section). Accordingly, we have been developing heuristic alternatives to exact evaluation, and comparing the results with optimal policies.

4 Optimal Deliberation Scheduling Agent

The Optimal MDP Agentdetermines the optimal decision to make in each possible state by employing an algorithm called *value iteration*. The basic idea of value iteration is to calculate the utility of each state by using the utilities of its successor states, and iterating over all states until the policy stabilizes. The Optimal MDP Agent constructs this optimal policy once, and then uses it repeatedly to take action, since the policy accounts for any possible state the agent might encounter.

Unfortunately, value iteration is only a feasible approach when the MDP model is very small. Figure 4 illustrates how quickly the number of states makes the computation intractable even for very simple MDP representations. This graph show the number of unique states generated by the Optimal MDP Agent for a mission of three phases, where the number of quanta per phase and the number of actions per phase are allowed to vary from 1 to 4.

Figure 5 similarly demonstrates how the computation time to generate the optimal policy for the set of states increases exponentially with small increases in problem size.

It should be noted that the results shown here assume that the Optimal MDP Agent need only generate the optimal policy once. However, if the world dynamics cause the deliberation management problem to change dramatically at any time, the entire policy would need to be recomputed, levying a huge penalty on this approach, and making the optimal MDP policy computation is even worse than might initially be apparent. This is not a problem the greedy approach we propose, and which we now discuss.

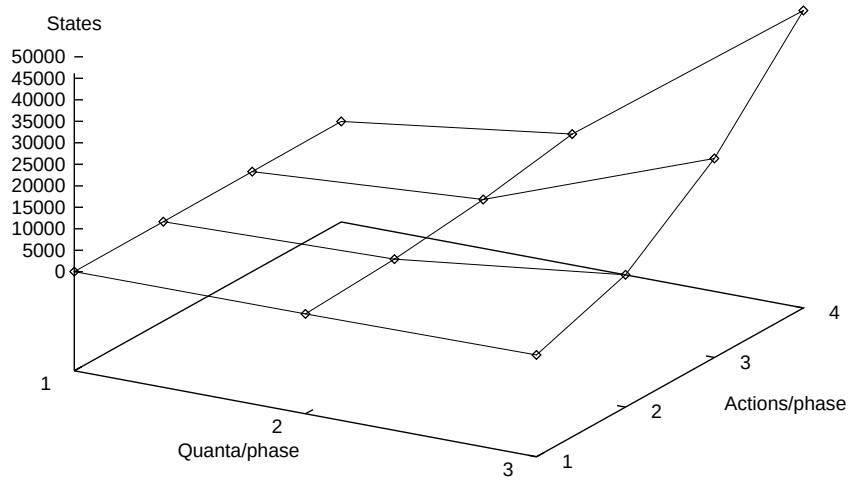

Figure 4: Number of states generated for simple MDP models.

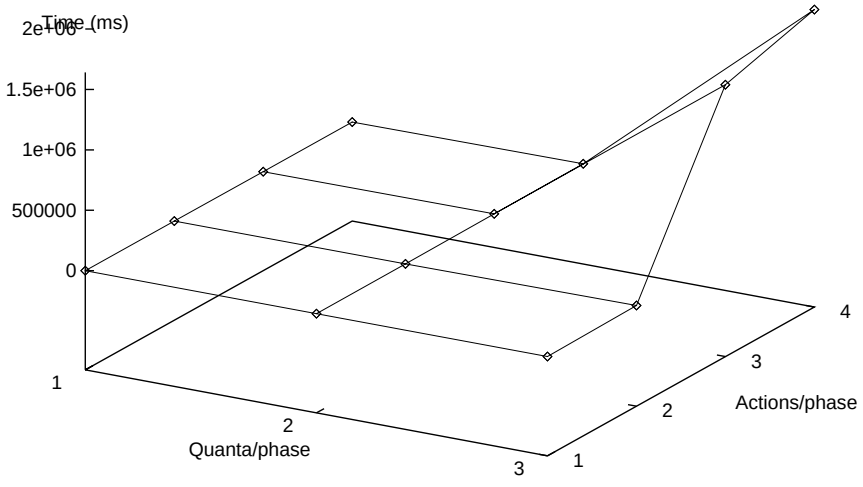

Figure 5: Time for MDP solver to find optimal policy: note huge Y axis scale.

5 Greedy Deliberation Management Agent

Because the optimal solution to our deliberation scheduling problem quickly becomes intractable, we have investigated a much simpler heuristic mechanism for making dynamic deliberation scheduling decisions. Rather than computing a policy that indicates what actions should be taken in any possible future state to maximize expected utility, the "greedy" agent myopically looks one state ahead along all of its immediate action choices and selects the action that results in the state (mission plan) with the highest expected utility. Since this expected utility computation is necessarily incomplete (because it does not project all future possible paths), the greedy agent is scalable but suboptimal.

Formally, we can say that the greedy policy is expressed in terms of a greedy utility measure $\overline{U}^{(1)}$.

$$\text{policy}^G(s) = \arg\max_{i,j} \sum_{s'} P(s \overset{m_j^{(i)}}{\rightarrow} s')\overline{U}^{(1)}(s')$$

The conventional utility measure used in the MDP formulation (see Section 3) must take an expectation over the future states, which have different plans. The myopic utility function, on the other hand, ignores all future planning, and assumes that the agent will complete the rest of the mission using the next-generated plan. That is, the greedy utility of state $s' = \langle t, b_i, P^t \rangle$ is defined as follows:[2]

$$\overline{U}^{(1)}(t, b_i, P^t) = (1 - p^t_{b_i})U(\text{destroyed}) + p^t_{b_i}(\delta_{t=\text{end}_{(i)}}(t)U(i) + \overline{U}^{(1)}(t+1, \text{phase}(t+1), P^t)$$

The greedy agent need not compute a complete policy; instead, it computes the policy lazily, determining the action choice for each state only when queried. Because the computation is local, its speed does not depend on the overall size of the MDP. In fact, it scales linearly with the branching factors of the MDP: the number of quanta per phase and the number of alternative operators per quantum. Figure 6 illustrates the runtime difference between the optimal and greedy agents. The first time the optimal agent is called upon to make a decision, it computes a complete policy and from then on it simply looks up its answer in the resulting hash table. The greedy agent performs its policy^G computation each time it is asked for a decision for a state.

The price for this efficiency is lack of optimality: because the greedy agent is making decisions with limited lookahead, it has trouble assessing the relative merit of addressing near-term vs. far-term risks. For example, it is easy to construct MDP problems with non-monotonic reward profiles that fool the greedy policy into focusing attention on later phases, causing it to miss near-term opportunities for improvement that could be addressed without precluding the later-phase improvements. The "pothole" example illustrated in Figure 7 illustrates a simple MDP survival probability distribution that dips early in the mission and then falls dramatically later. When assessing the possible actions at an early time in this mission, the simplest greedy policy will recognize the potential for large and important gains in survival probability in the last phase, and will choose to improve that phase first. The optimal policy, on the other hand, will recognize that it

[2] $\delta_{t=\text{end}_{(i)}}$ is a Kronecker delta function, valued 1 if its argument meets the condition, i.e., when t is the last time point in phase i, and zero everywhere else.

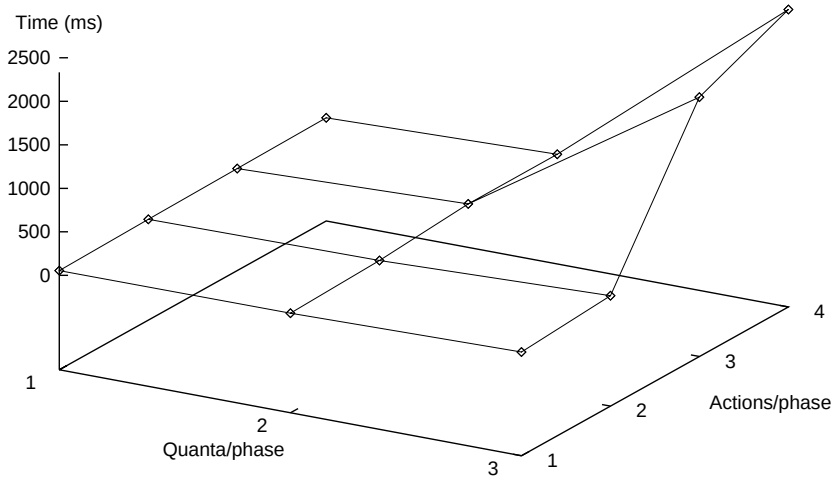

Figure 6: The greedy agent's runtime performance scales linearly, while the optimal agent is exponential. Note the small Y axis scale, as compared to Figure 5.

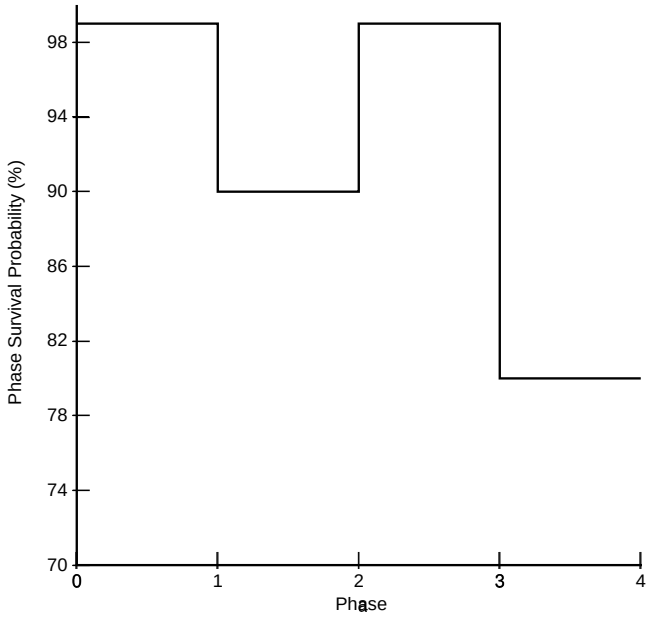

Figure 7: A non-monotonic initial survival probability distribution that fools a simple greedy heuristic.

has plenty of time during the mission to improve that phase, while the nearer-term dip in survival probability must be addressed quickly to have any effect. Indeed, for this scenario, the expected utility of the optimal policy computed by the MDP is approximately 0.85, while the expected utility of the greedy agent is only 0.63 (utility is normalized to a zero-one range).

To minimize the greedy agent's susceptibility to this effect, we can apply a discounting factor to future improvements. The discount factor captures the fact that there will be more opportunities in the future to perform plan improvements on mission phases that are farther in the future. Assuming that the discounting factor is α, a discounted myopic utility estimate ($\overline{U}_{\alpha}^{(1)}$) is as follows:

$$\overline{U}_{\alpha}^{(1)}(t, b_i, P^t) = (1 - p_{b_i}^t)U(\text{destroyed}) + p_{b_i}^t(\delta_{t=\text{end}_{(i)}}(t)U(i) + \alpha\overline{U}_{\alpha}^{(1)}(t+1, \text{phase}(t+1), P^t)$$

Figure 8: Results of experiments comparing optimal, greedy and time-discounted greedy agents.

The addition of the discounting factor adds only minimally to the cost of computing the greedy action selection, yet improves the performance of the agent considerably. For the "pothole" scenario, with $\alpha = 0.99$, the time-discounted greedy agent's performance is as good as the optimal agent. We conducted a number of experiments on randomly-generated domains, with four mission phases, with initial plans whose survival probabilities for each phase were drawn from a uniform distribution ranging from 80-100%. A plot of the results is given in Figure 8. The time-discounted greedy agent with $\alpha = 0.99$ performed better than the simple greedy agent in 175 of 287 experiments. Its average loss from the optimal policy was 11% and the average loss of the simple greedy agent was 15.2%.

6 Related Work

6.1 Anytime Algorithms

To the best of our knowledge, the term *deliberation scheduling* was introduced by Boddy and Dean. There was a great deal of work in this area around the early 1990s by Boddy and Dean [1],

Horvitz [4, 3], and Russell and Wefald [10]. These researchers (and others) investigated methods for using decision theory to address the problem of managing computation under bounded resources.

Boddy and Dean [1] categorize deliberation scheduling as being of two sorts: either discrete or anytime. In the discrete case, the agent must choose only *what* procedures to run, since the procedures are assumed to be uninterruptible and their run-times fixed. In the anytime case, procedures are assumed to be continuously-interruptible (anytime), and the agent must choose not only what procedure to run but also *when* and *for how long*.

Boddy and Dean and Horvitz' analyses and prescriptions are not *directly* applicable to the CIRCA deliberation scheduling problem, because they are particularly designed for controlling inference of suites of anytime algorithms. The CIRCA State-Space Planner is not well-suited to treatment as an anytime algorithm because it does not result in relatively continuous, smooth improvement in plans over time. Instead, the CSM acts more as a "batch mode" computation, taking in a particular problem configuration and, in general, returning either a successful plan (controller) or failure, after some amount of time.

6.2 Rational Metareasoning

Thus our work more closely fits the model proposed by Russell and Wefald (R & W) [10], in being concerned with discrete units of computation, rather than anytime algorithms. Russell and Wefald provide a framework for what they call "Rational Metareasoning." This framework centers around the notion of an agent that is continually trading off between performing some (atomic) computation, and executing a "real" action that will affect its environment.

In this framework, when evaluating a computation, R & W treat it as having two effects on utility. First, it causes time to pass, and thus can incur an opportunity cost. Specifically, it will cause the agent to postpone taking its next "real" action for at least the duration of one computational step. Second, a computation will have some effect on the real actions chosen by the agent. That is, there are two possible outcomes of a computation. The simpler one is that it may change what the agent believes to be the best available action. The more difficult to analyze second case is where the computation does not actually cause a change in action choice, but adds some information to the agent's state. That additional information, in turn, would cause *later* computation to change action choice. We can call this the indirect utility of computation.

Our work fits R & W's framework for analyzing the benefits of atomic computation actions. Our work differs in three important ways. First, we draw on the structure of CIRCA domains to make a specific, survival-oriented utility function, which is relatively easy to compute. Second, our simplified problem does not involve the indirect utility of computation discussed above. In the current model, a quantum of computation will either provide a direct utility, in the form of a new state space plan for a particular phase, or it will produce nothing at all.

The third difference between our work and R & W's is that we are not faced with the problem of trading off deliberation versus action. Because CIRCA was designed to act in hard real-time environments, the CIRCA execution engine (the Real-Time Subsystem) was designed to execute in parallel with the AI subsystem and not to compete for computational resources [6]. So CIRCA deliberation scheduling involves only tradeoffs between alternative *planning* activities, not between planning and action.

The first two simplifications mean that it is trivial to estimate the value of a single quantum of computation in our model. The simplifications, however, impose a cost on the performance of our agents. In particular, it is never possible for our agents to gain the benefit of any computation that takes more than a single quantum of time. Future work will be aimed at relaxing this restriction. This relaxation will correspond to allowing the AMP to "resume" or "continue" controller synthesis tasks that it has already started in a prior deliberation quantum.

6.3 Design to Criteria

Another closely related stream of research includes the "design-to-time" (DTT) [2] and "design to criteria" (DTC) [11] efforts, which consider agent deliberation scheduling in the context of TAEMS task models. These task models capture some of the characteristics of our AMP problem (e.g., alternative atomic deliberation actions, success probabilities) and also can include more complex aspects such as task decomposition, interdependencies, and resource constraints. The DTT and DTC techniques use heuristics to build satisficing schedules for a full sequence of future activities; in contrast, our current approach to the AMP deliberation scheduling problem has emphasized bounded-time methods to return a single decision about the next deliberation action to take. In principle, we could model our problem in TAEMS and compare the current DTC system against our more myopic heuristics. If the software is available and compatible, this may prove a profitable avenue for future investigation.

6.4 Future Work

In this paper, we concentrate our attention on the AMP's management of the Controller Synthesis Module's deliberation. However, in general, we plan to endow the AMP with broad power and responsibility, requiring more extensive and sophisticated reasoning capabilities which intelligently consider more aspects of realistic problem domains. As examples, we plan to expand the set of techniques to adjust the CSM problem complexity by explicitly bringing goals under control, enable agents to negotiate to intelligently off-load burdens to other agents, and allow the AMP to add and remove control actions from the set available to each agent (for example, to make a plan feasible, or to force a plan to be more efficient).

Even within the scope of deliberation management, we plan to incorporate a much richer set of problem configuration modification operators with varying characteristics. For example:

Time Horizons: In many cases, the AMP can simplify a CSM planning problem by exploiting the observation that the resulting plan only needs to guarantee safety for a bounded time period (i.e., up to a *horizon*). For example, if the AMP knows that it already has (at least) a default plan to switch to after a certain length of time spent in the current plan, the CSM need not plan for any states that cannot possibly be reached within that horizon. We call the horizon *admissable* if the AMP can *guarantee* that it will swap in a new plan within the horizon (e.g., if an action to transition into the new plan will definitely be taken within some time bound). If the horizon is admissable, then imposing a time horizon on the CSM planning does not compromise safety, and the improvement is an optimization, rather than a tradeoff. The inadmissable case is a tradeoff with some likelihood of finding the agent in an unhandled state (beyond the horizon).

Timing Modifications on Temporal Transitions: CIRCA has traditionally always considered the worst-case time for temporal transitions to move the world from state to state. However, in overconstrained domains, this assumption can lead the CSM to conclude there is no safe plan. In situations where the likelihood of temporally transitioning at the earliest possible time is very low, a tradeoff can be made to modify (lengthen, in this case) the temporal transition time. Of course, if the transition occurs outside of the modified interval, safety is not guaranteed.

Minimal Regions of Competence: In the problem configuration, the AMP can specify how "goal-achieving" it wants the CSM to be as it generates a plan. Then, as the CSM is faced with decisions over which action to choose in each state, it can choose actions that increase the probability of goal achievement, or prefer to "bend back" into a state that has already been generated (and handled). The latter preference decreases the complexity of the planning problem by minimizing the set of states the CSM must plan for. In general, safety is not compromised with this method, but goal achievement might be.

Systematic Search for Feasible Problem: The AMP must know what to do when the CSM returns with no feasible controller, or does not return a controller in an acceptable amount of time. One solution is to have the AMP heuristically decide how to simplify the problem configuration in a systematic way, essentially searching for the highest-utility problem configuration for which the CSM can find a feasible plan.

Acknowledgments

This material is based upon work supported by DARPA/ITO and the Air Force Research Laboratory under Contract No. F30602-00-C-0017.

Thanks to Mark S. Boddy for guiding us gently into the literature on deliberation scheduling. Any misprision is entirely our own fault.

Thanks to Steven A. Harp for advice on experimental design and statistical hypothesis testing.

References

[1] M. Boddy and T. Dean, "Decision-Theoretic Deliberation Scheduling for Problem Solving in Time-Constrained Environments," *Artificial Intelligence*, 1994.

[2] A. Garvey and V. Lesser, "Design-to-time Real-Time Scheduling," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 23, no. 6, pp. 1491–1502, 1993.

[3] E. Horvitz, G. Cooper, and D. Heckerman, "Reflection and action under scarce resources: Theoretical principles and empirical study," in *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pp. 1121–1128. Morgan Kaufmann Publishers, Inc., 1989.

[4] E. J. Horvitz, "Reasoning under varying and uncertain resource constraints," in *Proceedings of the Seventh National Conference on Artificial Intelligence*, pp. 111–116, Los Altos, CA, 1988, Morgan Kaufmann Publishers, Inc.

[5] D. J. Musliner, "Imposing Real-Time Constraints on Self-Adaptive Controller Synthesis," in *Lecture Notes in Computer Science*, number 1936, Springer-Verlag, 2001.

[6] D. J. Musliner, E. H. Durfee, and K. G. Shin, "CIRCA: A Cooperative Intelligent Real-Time Control Architecture," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 23, no. 6, pp. 1561–1574, 1993.

[7] D. J. Musliner, E. H. Durfee, and K. G. Shin, "World Modeling for the Dynamic Construction of Real-Time Control Plans," *Artificial Intelligence*, vol. 74, no. 1, pp. 83–127, March 1995.

[8] D. J. Musliner, R. P. Goldman, M. J. Pelican, and K. D. Krebsbach, "SA-CIRCA: Self-adaptive software for hard real time environments," *IEEE Intelligent Systems*, vol. 14, no. 4, pp. 23–29, July/August 1999.

[9] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, 1995.

[10] S. Russell and E. Wefald, "Principles of Metareasoning," in *First International Conference on Principles of Knowledge Representation and Reasoning*, pp. 400–411. Morgan Kaufmann Publishers, Inc., 1989.

[11] T. Wagner, A. Garvey, and V. Lesser, "Criteria-Directed Task Scheduling," *International Journal of Approximate Reasoning*, vol. 19, no. 1–2, pp. 91–118, 1998.