Flexibly Integrating Deliberation and Execution in Decision-Theoretic Agents

David J. Musliner, Jim Carciofini
Honeywell Laboratories
{david.musliner,jim.carciofini}@honeywell.com

Robert P. Goldman
SIFT, LLC
rpgoldman@sift.info

Edmund H. Durfee, Jianhui Wu
University of Michigan, Ann Arbor, MI
{durfee,jianhuiw}@umich.edu

Mark S. Boddy
Adventium Labs, Minneapolis, MN
mark.boddy@adventiumlabs.org

Abstract

We are developing software agents that plan, schedule, and coordinate complex behavior in uncertain environments by reasoning about dynamically-constructed Markov Decision Problems (MDPs). What makes the work we present here different from traditional MDP-based agent systems is that an agent in our system might lack the time and/or knowledge to build its complete MDP and corresponding optimal policy prior to when it must begin execution. We have developed several new techniques to handle this system-level embedding in a real-time environment while retaining the advantages of decision-theoretic reasoning. In this paper, we describe how our agent architecture smoothly combines dynamic expansion and solution of the (partial) MDP models with real-time execution of the policies being developed. This combination is very flexible, allowing the agent to deliberate whenever possible and leverage that deliberation in its near-term decisions. The combination is also synergistic: information about the asynchronous execution of tasks and their outcomes allows the agent to prune its deliberation about future possibilities. As a result, the agent can self-adapt to dynamically-arriving problems with wide variations in deliberation time, both before and during policy execution.

Introduction

We are developing distributed teams of embedded software agents that coordinate their behaviors dynamically by reasoning about complex, hierarchical, distributed task models. These agents may be used to coordinate teams of humans or autonomous systems engaged in a wide variety of complex real-time coordinated activities (e.g., search-and-rescue, military operations, refinery control, distributed web services). The agents reason about task models expressed in the C-TÆMS language, which includes explicit uncertainty about the durations and outcomes of tasks. Our agents reason explicitly about this uncertainty by each casting its local planning and scheduling problem as the solution to a Markov Decision Problem (MDP).

The agents operate in a time-constrained environment where each generally cannot build its entire MDP and corresponding action policy before it must begin executing tasks. Furthermore, an agent's goals and task models can change on the fly, and it must dynamically adapt its behavior accordingly. In this paper, we describe how our agent architecture smoothly combines continuous expansion and solution of the (partial) MDP models with real-time execution of the policies being developed. This combination is very flexible, allowing the agent to deliberate not only before but during execution, leveraging its ongoing deliberation to improve its near-term decisions. In addition, the agent can use information about the outcomes of tasks that it had previously decided to execute in order to prune from consideration state trajectories that are now impossible. As a result, the agent can self-adapt to problems with wide variations in deliberation time both before and during execution.

In the next section we briefly introduce the C-TÆMS task modeling language and the associated simulation environment that define the problems and real-time execution domains our agents inhabit. C-TÆMS task models can be interpreted as a relatively compact representation for a large multi-agent MDP state space. In the following section, we introduce our first contribution, "Informed Unrolling," which is a technique for dynamically guiding the anytime construction of this MDP from the C-TÆMS model. This technique is related to prior work on anytime MDP solving, but is particularly tailored to the requirements (and advantages) of operating in a real-time execution environment. We then discuss the interaction between the real-time execution of the (partial) MDP policy and the deliberation processing that is continuing to extend and revise the underlying MDP model. We conclude with a discussion of system-level results showing how the resulting agent can use the available reasoning time to its best advantage while also incorporating information from ongoing execution of its partial policies.

C-TÆMS: Multi-Agent Probabilistic Task Models

The C-TÆMS language (Boddy *et al.* 2005) represents multi-agent hierarchical tasks with stochastic outcomes and complex hard and soft interactions. Unlike other hierarchical task representations, C-TÆMS emphasizes complex reasoning about the utilities of tasks, rather than emphasizing interactions between agents and the state of their environment.

C-TÆMS permits a modeler to describe hierarchically-structured tasks executed by multiple agents. A C-TÆMS task network has *nodes* representing *tasks* (complex actions) and *methods* (primitives). Nodes are temporally extended:

they have durations (which may vary probabilistically), and may be constrained by release times (earliest possible starts) and deadlines. Method executions that violate the temporal constraints yield zero quality (and are said to have *failed*). At any time, each C-TÆMS agent can be executing at most one of its methods, and no method can be executed more than once.

A C-TÆMS model is a discrete stochastic model: methods have multiple possible outcomes. Outcomes dictate the *duration* of the method (an integer greater than zero) and its resulting *quality* (effectively a non-normalized utility measure).

Every task in the hierarchy has associated with it a *quality accumulation function* (QAF) that describes how the quality of its children is aggregated up the hierarchy. The QAFs combine both logical constraints on subtask execution and how quality accumulates. For example, a :MIN QAF specifies that all subtasks must be executed and must achieve some non-zero quality in order for the task itself to achieve non-zero quality, and the quality it achieves is equal to the minimum achieved by its subtasks. The :SYNCSUM QAF is an even more interesting case. Designed to capture one form of synchronization across agents, a :SYNCSUM task achieves quality that is the sum of all of its subtasks that start at the same time the earliest subtask starts. Any subtasks that start after the first one(s) cannot contribute quality to the parent task.

The quality of a given execution of a C-TÆMS task network is the quality the execution assigns to the (unique) root node of the task network. C-TÆMS task networks are required to have deadlines on their root nodes, so the notion of the end of a trace is well-defined.

Traditional planning languages model interactions between agents and the state of the environment through preconditions and postconditions. In contrast, C-TÆMS does not model environmental state change at all: the only thing that changes state is the task network. Without a notion of environment state, in C-TÆMS task interactions are modeled by *non-local effect* (NLE) links indicating inter-node relationships such as enablement, disablement, facilitation, and hindrance.

Figure 1 illustrates a simple version of a two-agent hostage-rescue scenario. The whole diagram shows a global "objective" view of the problem, capturing primitive methods that can be executed by different agents (A and B). The COORDINATORS agents are *not* given this view. Instead, each is given a (typically) incomplete "subjective" view corresponding to what that individual agent would be aware of in the overall problem. The subjective view specifies a subset of the overall C-TÆMS problem, corresponding to the parts of the problem that the local agent can directly contribute to (e.g., a method the agent can execute or can enable for another agent) or that the local agent is directly affected by (e.g., a task that another agent can execute to enable one of the local agent's tasks). In Figure 1, the unshaded boxes indicate the subjective view of agent-A, who can perform the primitive methods Move-into-Position-A and Engage-A. The "enable" link indicates a non-local effect dictating that the Move-into-Position-A method must be completed successfully before the agent can begin the Engage-A method. The diagram also illustrates that methods may have stochastic expected outcomes; for example, agent-B's Move-into-Position-B method has a 40% chance of taking 25 time units and a 60% chance of taking 35 time units. The :SYNCSUM QAF on the Engage task encourages the agents to perform their subtasks starting at the same time (to retain the element of surprise).

One of the defining features of C-TÆMS problems is the degree to which the agents are *embedded and online*, meaning that the time required for deliberation takes time in the real (or simulated) world of execution, and that the situation can change during execution in ways that are beyond the agents' control and that require replanning. In addition to the modeled uncertainty exemplified above, C-TÆMS problems can also include dynamically-arriving task hierarchies which are presented to the executing agents during a scenario, while they are already executing other tasks. New C-TÆMS task models may cause an agent to abort its ongoing tasks, replan in part or whole, or do nothing different. Similarly, C-TÆMS can express dynamically-arriving changes to existing model characteristics (e.g., the deadline of a node may change after it has already been reasoned about). Robustly handling all these forms of real-time environmental dynamics and uncertainty is a key requirement for our agents.

Partial MDPs: "Informed Unrolling"

We refer to the process of converting a C-TÆMS problem into an MDP problem as "unrolling," because it involves projecting forward from an initial state (where no methods have been executed) to imagine future possible states of the C-TÆMS network in which some methods have been chosen for execution at particular times and have received particular outcomes. The core unrolling algorithm is thus a simple state-space enumeration process where an MDP state is expanded by creating the successor states that result from each of the possible action choices and their outcomes. These successor states are added to an *openlist* of un-expanded states, and the process ideally continues until the openlist is empty and the full reachable MDP state space has been enumerated.

Since full enumeration of even single-agent C-TÆMS MDPs is often impractical, we have developed a technique for heuristically-guiding the enumeration of the reachable MDP state space to include in the MDP the states most relevant for finding the optimal policy. Our *informed unroller* (IU) algorithm prioritizes (sorts) the openlist of states waiting to be unrolled based on an estimate of the likelihood that the state would be encountered when executing the optimal policy from the initial state.

One cannot determine the probability of reaching a state without considering the policy followed by the agent in previous states. Therefore, the IU intersperses policy-formulation (using the Bellman backup algorithm) with unrolling. This means that we must be able to find an (approximately) optimal policy for partial MDP state spaces, which means we must have a heuristic to use to assign a quality estimate to a current leaf node to represent the reward that

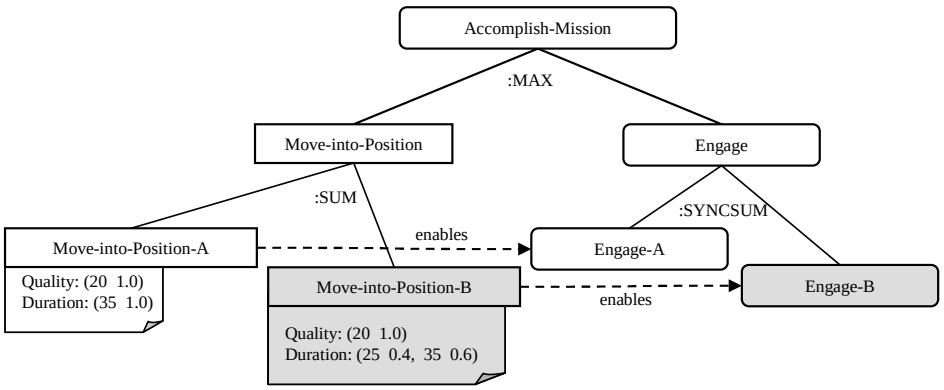

Figure 1: A simple C-TÆMS task network for two agents, illustrating some of the representation features. Some details have been omitted for brevity.

would be gained if the agent were to reach that node and execute an optimal policy from there. We have developed a suite of alternative heuristics for estimating intermediate state quality, since the problem of finding a good heuristic is quite difficult (Wu & Durfee 2007).

One heuristic approach that often strikes a good balance between computation time and accuracy is to do a greedy forward search from the intermediate state to a terminal state (at the deadline of the overall top-level task). The agent finds the method to execute in the intermediate state that will yield the best immediate next expected reward, and the finds the best method from the most likely state that would be reached by taking that method, and so on, to create in a greedy manner a simple linear plan representing one possible good trajectory to a terminal state. The utility of reaching that state is then used for the estimated utility of the intermediate state when computing the policy for the partially-unrolled space.

Because computing the probabilities of reaching the edge states of the partially-unrolled MDP requires performing a Bellman backup on that state space, sorting the openlist (which can be quite large) for the IU can be an expensive operation. Therefore we constrain when the openlist sorting takes place by using an active meta-control function that tries to decide when to start a sort (and the associated policy derivation) based on an estimate of the expected sort time and consideration of the upcoming real-time deadlines (e.g., the next time an action decision is required). This has the net effect of sorting the openlist more often early in the search, when focus is particularly important, and less often or never as the space is unrolled farther and probability information becomes both less discriminatory (because the probability mass is distributed over a very large set of reachable edge nodes) and focus becomes less critical (because the agent has time to refine its model/policy before reaching those states).

Early results from our evaluation of the IU algorithm against a complete solution of (small) MDPs are promising. For example, in Figure 2 we show a comparison of the performance of the informed unroller against the complete unrolling process. In these small test problems, the informed unroller is able to find a high-quality policy quickly and to

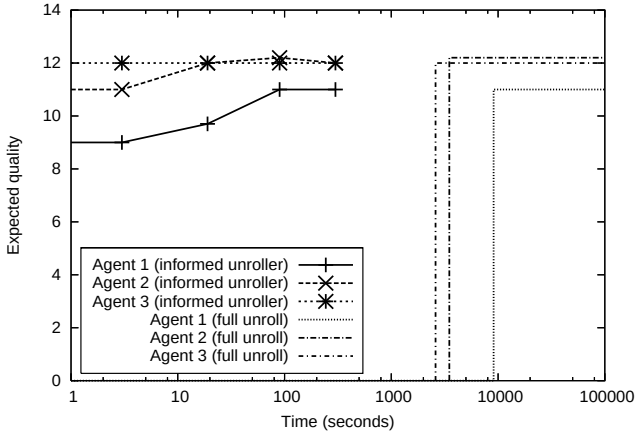

Figure 2: The Informed Unroller can often find near-optimal policies much faster than building the complete MDP.

return increasingly effective policies given more time. This allows the IU-agent to flexibly trade off the quality and timeliness of its policies.

The IU method is a special case of the find-and-revise algorithm schema (Bonet & Geffner 2006) (which is a generalization of algorithms such as LAO^* (Hansen & Zilberstein 2001)). These algorithms use knowledge of the initial state(s) and heuristics to generate a state subspace from which a policy can be extracted. Our technique differs from the general case, and its instances, in substantial ways. LAO^* generates a state subspace from which the optimal policy can be provably derived. The IU, on the other hand, executes online, and might lack enough time to enumerate such a state subspace even if it knew exactly which states to include. The IU is an anytime algorithm and is integrated into an execution environment, unlike LAO^*, which runs offline. For this reason, the IU makes no claims about policy optimality; indeed, it is not even guaranteed to generate a closed policy.

The general find-and-revise algorithm family can provide

guarantees weaker than those of LAO^*, but those guarantees rely on having an admissible heuristic value function for states that have not been fully explored. However, even if we had an admissible heuristic, it is not at all clear that the IU should use it. An admissible heuristic will tend to push the policy expansion to explore states where it is **possible** that the optimum will be found, in order that we not miss the optimum. However, the IU is operating in a time-pressured domain. So we should not necessarily be encouraging the system to move toward promising unexplored areas — that will tend to leave the agent with a policy that is broad but shallow, and thus make it more likely that it will "fall off policy" during execution. Instead of admissibility, we must find a heuristic function that will cause the agent to tend to build policies that trade off considerations of optimal choice against completeness/robustness of the policy. It is possible that this heuristic should be time-dependent — as the agent runs out of time for policy development, the IU's heuristic should focus more on robustness and less on optimality.

Executing the MDP Policy

Once a partial MDP policy has been derived, the system is ready to begin executing that policy as necessary. Problems are posed to the system with a limited amount of deliberation time before the first method can be executed (typically, 20 seconds to 5 minutes). Each time the IU algorithm performs an openlist sort, it recomputes the optimal policy for the larger state space (to drive the determination of the probabilities for leaf nodes in the partial MDP based on that policy), so as soon as the first openlist sort has completed, the agent is ready to execute its current best policy. Each time the openlist is sorted, the current best policy is overwritten.

To execute the policy, the agent keeps track of the current state in the MDP that best represents what has happened so far in the world outside the agent (currently, a simulation). Beginning with the initial state, in which no methods have been executed, the system chooses to execute a method based on the current best policy, choosing the method that leads to the state with the highest expected utility.

Once it has started a method, the executive listens for information from the (simulated) environment telling it what outcome the method achieved. The executive determines, from that information, which transition in its MDP was actually traversed, and so what its new current-state is. The agent consults its current best policy to select the action to take in that state, and this process repeats as long as the agent's MDP is sufficiently complete and correct, in that the agent has been able to anticipate and plan for each state by the time the state is reached. Unfortunately, this is not always the case.

The MDP might not be sufficiently complete, because the informed unroller is always fighting against time. As execution progresses, the agent can "catch up" to the edges of the unrolled space, and so the executive can reach a state that the IU has not yet unrolled, or a state that has been unrolled but was not in the MDP during the most recent policy-generation and thus does not have an action planned for it. When this happens, the agent can no longer follow its policy; it falls "off-policy" and enters an off-policy execution mode, as will be described shortly.

The MDP might not be sufficiently correct if the local C-TÆMSmodel used by the IU is incomplete because it lacks information about all the possible non-local effects (NLEs) of other agents. For example, a method may be *facilitated* by another method, either done locally or by another agent. If the source task of the facilitation earns quality before the target method is executed, then the method's outcome will not follow its normal modeled outcome behavior, but will instead be both faster and of higher quality, depending on the quality achieved at the NLE source.

For methods that are facilitated by purely local (same-agent) tasks, it is possible to enumerate all the possible facilitated outcome values and thus build a complete MDP through which the executive can track precisely. More generally, for distributed problems and problems that have not been unrolled entirely, an agent might receive back from the (simulated) environment information about the method that it is executing that does not match any of the MDP transitions associated with the method. For example, suppose that a method is facilitated by a non-local method and this results in an outcome that is 10% faster and 10% higher-quality than the un-facilitated method can achieve. If the agent has no specific model of this facilitation, its MDP will not contain states with those improved values, and the executive's state-tracker could fall into an off-policy execution mode.

To enable agents to remain on-policy through such events, our executive tolerates some amount of mismatch during state tracking when method outcomes are better (higher quality and/or faster) than expected. When an exact transition cannot be found, the executive tracks to the modeled state that is closest in time and quality to what occurred. We say in this case that the agent *warps* to a nearby modeled state, and then follows its policy as if that warped-to state were the real state. Because the time at which the warped-to state was expected to be reached might be later than the current time, the warping process can include the insertion of a no-op (WAIT) method until the times align. From that point on, the MDP will have a feasible, if imperfect, set of expectations, relying only on lower expected quality than that which was actually achieved.

There are many situations in which even this warping behavior is not sufficient to keep the agent on-policy, and the executive realizes that the MDP it has unrolled and planned for is no longer well aligned with the environment. For example, the C-TÆMS language allows *meta-TÆMS* model changes during execution, including changes to temporal constraints like deadlines, changes to expected outcome distributions, and even the arrival of entirely new portions of the problem domain (new hierarchical tasks, new methods). When a *meta-TÆMS* event occurs in the midst of execution, the agent is in a tough situation: using its current MDP could be no longer appropriate, but stopping taking actions while it computes a new MDP could cause it to fail to take critical actions at necessary times. Our agent has mechanisms in place that permit it to continue executing its old policy while deliberating about the new policy that it should follow, and

then later switching from the old to the new at an opportune time. There are many challenges in doing this correctly, and we leave the details of this process to a future paper.

Despite the efforts of the IU to generate useful policies quickly, and of the executive to follow the current best policy flexibly, an agent can sometimes fall off-policy. When that happens, the agent falls into one of several off-policy modes (which one is currently based on a user-settable flag). One of the most effective off-policy modes is to use the same greedy heuristic technique as is used to compute the heuristic value to assign to an edge state. That is, the executive looks at the expected next states given each of the currently applicable methods, and selects the method that will reach immediate successor states with the highest-expected utilities.

Synergy Between Execution and Deliberation

A crucial advantage of our approach to executing the partial MDP's policy while we continue to unroll more of the MDP is that not only can an agent use the results of deliberation (improving the policy) to inform execution (choosing the next method to execute), but it can also use the results of execution to inform deliberation. Recall that an agent can fall off-policy when execution catches up with the edges of the unrolled state space, such that the policy does not cover the state reached. Execution can generally catch up because of the branching factors involved: the width of the expanded state space keeps growing exponentially given the branching due to the method choices and their stochastic outcomes.

But once execution has begun, and the actual outcomes of executed methods are known, then the space of reachable states can be narrowed down. For example, as soon as the system has chosen to execute a particular method in a state, no other choice is possible. The other branches of the MDP out of the current state can be pruned, and huge portions of the subsequent state space may become unreachable.

There are three kinds of pruning opportunities that occur during execution of a single-agent problem:

- **Action choices —** As above, when the executive chooses to start a method in a state, all other actions (method choices or WAITs) are no longer possible, so all their transitions are removed from the MDP.
- **Outcomes that occur —** When the executive learns that a method has completed at a certain time and yielded a certain quality, that information is used to update the current state in the MDP. Only states reachable from the current state are still of interest to the deliberation process. This type of pruning can be easily accomplished either by moving the initial state of the entire MDP to the new current state of the agent, or by pruning all other transitions out of the prior state.
- **Outcomes that do not occur —** When the executive gets a pulse message indicating that a new discrete time tick is starting but the currently-executing method did not complete on the prior tick, this itself can provide pruning information. If one of the outcomes of the executing method should have finished in the prior tick, then the fact that it did not finish is sufficient to prune that outcome's transition from the MDP.

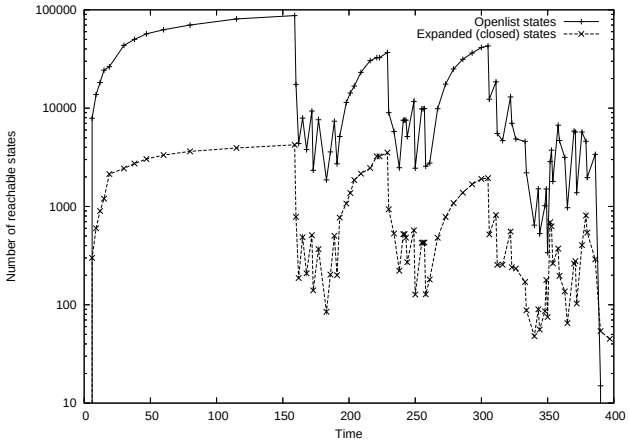

Figure 3: Execution-time state-space pruning periodically reduces the reachable state-space dramatically, while the Informed Unroller continually expands the projection of future states.

Because the executive runs asynchronously from the deliberation engine (MDP unroller), the execution-based pruning must be implemented carefully to avoid corrupting the evolving MDP model. In our case, as the executive encounters the pruning opportunities above, it queues up a list of transitions to be pruned. At the top of the MDP unrolling loop, when the MDP model is stable, the queued pruning opportunities are processed to remove transitions from the MDP. If any prunes are made, a topological sort algorithm is run on the MDP, deriving the revised reachable state space, the new openlist, and the topology for Bellman backup. Then the openlist popping and state expansion proceeds as usual.

Finally, it should be noted that even when *meta-TÆMS* events occur, pruning can still be exploited. When the MDP is being formed for the improved model of the agent's tasks, the IU can begin with the state that has actually been reached so far. Then, as new states are reached and methods are chosen from the current policy, the effects of these choices can be mapped across to isomorphic components of the new model being unrolled and reasoned about. In this way, an effective policy for the new model can be more rapidly produced and swapped in.

Results

By integrating execution and deliberation over a continuously-expanding partial MDP policy, our CO-ORDINATOR agents are able to take efficient advantage of all the deliberation time they are given. When the domain offers a relatively long deliberation period, the agents may unroll a very extensive MDP, sometimes even completely considering all possible futures and generating a complete optimal policy (modulo possible imperfections in the inter-agent agreements). When less time is available, they generate intermediate partial policies that are approximately optimal for the state space considered so far. And, when execution-time information can reduce their uncertainty

about possible future states, the agents aggressively prune the partial MDP model, focusing their deliberation only on reachable future states.

For example, Figure 3 illustrates the MDP size over time for a single agent operating in a domain where the first method could be started after 140 seconds, and thereafter the different methods being selected took 3 to 10 seconds to execute. Adapting to the long deliberation period at the start, the agent unrolled a state space of nearly 100,000 states before it could execute its first action. That first action choice caused a huge pruning effect, reducing the explored reachable state space to only a few thousand states. Then, as the first method executed for a few seconds, the unroller expanded its projections about the future, made another action choice which reduced the reachable state space, etc. Several times in this problem the domain made it impossible to run any methods (e.g., from time 200 to 240) and the agent used that time to dramatically expand its future planned state space. So the agent dynamically adapts it deliberation to the available time, and uses any conclusions of its deliberations immediately during the concurrent problem execution.

Combined with execution-time state warping, these features enable our agents to flexibly self-adapt to domains with widely varying real-time responsiveness requirements and extremely large possible state spaces, remaining in control and on-policy (within the portion of the state space they have considered) despite the uncertainty of their actions.

Future Work

In this paper, we concentrated on the deliberation and execution activities within an agent, and how they can work off of each other to the agent's benefit, but there are many other issues that arise in developing techniques for the kinds of complex, realistic applications that we are facing. Some of these deal with making the deliberation more far-sighted; a major benefit of MDPs is modeling how downstream eventualities should influence earlier decisions, and that benefit is compromised in our time-constrained problems where the MDP can only be partially unrolled. Strategies such as phasing (Wu & Durfee 2007) might help address these limitations.

Similarly, we have not gone into detail in this paper about how these techniques need to be adjusted for multiagent contexts, such that the decisions made about what portions of the MDPs to deliberate about and where to exercise execution flexibility are made in a coordinated manner. These can be especially important if we find other ways to keep the agent on-policy despite minor unmodeled perturbations in outcomes by using even more flexible warping, mapping the actual current state to an unrolled state that is not a strictly admissible approximation. This would allow the agent to remain on-policy and within its partially-optimized plan, instead of falling off-policy and operating in a myopic/greedy mode, but if different agents exercise flexibility differently, the policies that they end up following might be mismatched and their collective performance might suffer.

In summary, while MDP solution methods have been studied quite widely, there is very little literature on making that solution process real-time and responsive to an executing, uncertain, changing environment. As MDP-solution techniques scale up to handle problems of practical size (e.g., military unit planning and coordination), the problems of real-time performance and execution will be increasingly in the spotlight. Our IU-agent is breaking new ground in self-adaptive deliberation and integration with execution, in the context of highly uncertain and variable task models.

Acknowledgments

This material is based upon work supported by the DARPA/IPTO COORDINATORS program and the Air Force Research Laboratory under Contract No. FA8750–05–C–0030. Any opinions, findings and conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA, the U.S. Government, or the Air Force Research Laboratory.

References

Boddy, M.; Horling, B.; Phelps, J.; Goldman, R. P.; and Vincent, R. 2005. C-TÆMS language specification. Unpublished; available from this paper's authors.

Bonet, B., and Geffner, H. 2006. Learning depth-first search: A unified approach to heuristic search in deterministic and non-deterministic settings, and its application to MDPs. In Long, D.; Smith, S. F.; Borrajo, D.; and McCluskey, L., eds., *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling*, 142–151.

Hansen, E. A., and Zilberstein, S. 2001. LAO: a heuristic search algorithm that finds solutions with loops. *Artificial Intelligence* 129(1-2):35–62.

Wu, J., and Durfee, E. H. 2007. Solving large taems problems efficiently by selective exploration and decomposition. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS07)*.