Working Notes of the AAAI Spring Symposium on Distributed Plan and Schedule Management
Stanford, CA, March 2006

Coordinated Plan Management Using Multiagent MDPs

David J. Musliner
Honeywell Laboratories
david.musliner@honeywell.com

Edmund H. Durfee, Jianhui Wu, Dmitri A. Dolgov
University of Michigan
durfee,jianhuiw,ddolgov@umich.edu

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

Mark S. Boddy
Adventium Labs
mark.boddy@adventiumlabs.org

Introduction

For the past several years, we have been developing multi-agent technology to help humans coordinate their activities in complex, dynamic environments. In recent work on the DARPA COORDINATORS program, we have developed multi-agent Markov-decision process (MDP) techniques for distributed plan management. The COORDINATORS problems arrive in distributed form, with different agents getting local views of their portion of the problem and its relationship to others. Even so, the individual agents' MDPs that capture their local planning and scheduling problem can be too large to enumerate and solve. Furthermore, the COORDINATOR agents must build and execute their plans in real-time, interacting with a world simulation that makes their actions have uncertain outcome.

Accordingly, we have developed an embedded agent system that negotiates to try to find approximately-optimal distributed policies within tight time constraints. Our work draws together and extends ideas in multi-agent Markov decision processes, real-time computing, negotiation, meta-level control, and distributed constraint optimization. Contributions of our work include "unrolling" techniques for translating local hierarchical task networks to MDPs, "informed" heuristic search control of the unrolling process, and negotiation methods for allocating responsibilities across cooperating agents and using those allocations to influence local policy construction.

In the rest of this paper, we describe our approach in more detail. We begin by summarizing the challenges in distributed plan management embodied in the COORDINATORS problem, and the TÆMS representation used to model the actions and interactions requiring coordination. We then describe how we translate the problem represented in TÆMS into an MDP, and the strategies that we use for finding policies when the MDP state space exceeds the time and/or space bounds for our system. After that, we discuss the challenge of using inter-agent negotiation to coordinate the agents' policies. Finally, we point out the limitations of our initial implementation, and we outline our plan to improve the management of uncertain and unexpected events by more fully integrating ongoing deliberation and coordination.

The COORDINATORS Problem

COORDINATORS is a research program funded by DARPA IPTO to identify, prototype, and evaluate well-founded technical approaches to scheduling and managing distributed activity plans in dynamic environments. As a motivating example, consider the following scenario. A hostage has been taken and might be held in one of two possible locations. Rescuing the hostage requires that both possible locations are entered by special forces *simultaneously*. As the activities to move personnel and materiel into place are pursued, delays can occur, or actions to achieve precursor objectives might have unexpected results (e.g., failure). COORDINATOR agent systems will be associated with the various human participants. COORDINATOR agents should monitor the distributed plans and manage them as the situation evolves, to increase their effectiveness and make them more likely to succeed.

In general, a set of COORDINATOR agents is meant to work together to maximize the reward gained by the group as a whole. In other words, the problem is to compute an effective *joint* policy for the agent society, in which the actions taken by one agent can depend on the state of the group as a whole, not just the local state of that agent. The agents are time-pressured: each agent must make timely action decisions during execution. Furthermore, the problem must be solved in a distributed fashion.

Each agent's partial model includes the actions that the agent can execute, which are stochastic, rather than deterministic, and some of the actions its peers can perform. The model also provides *partial* information about the rewards that the society as a whole will receive for reaching various states. This model is not static: the agent can receive model updates during execution. Therefore, agents must be able to manage and reformulate policies reactively.

The problems are formulated as complex hierarchical task networks, which we translate to MDPs. It is our hypothesis that MDPs provide the appropriate modeling tool for representing the COORDINATORS problem, and that each agent should generate, store, and follow the best policy that its knowledge and available resources (time, etc.) allow. To explore this approach, we must develop distributed negotiation techniques for coordinating the policy-finding activities of the various agents, and we must provide a *local* utility model that will cause the individual agents to (approximately) maximize group expected utility by (approximately) maximizing their local expected utility.

The intractable space of joint multiagent policies means that our negotiation protocols for collaboratively arriving at joint policies can explore only a small portion of the joint policy space. Our approach reduces this full space by projecting down to an alternative space of joint "commitments" which bias the agents' local policy formulation processes. However, because the MDPs are generally too large to create and solve within the environment's time limits, the agents can only formulate partial and approximate policies.We have developed single-agent policy-finding techniques that enable an agent to flexibly trade off the quality of a policy for time. At runtime, the agents monitor their changing local states, and model the changes to each other's states, tracking their passage through the MDP state space and taking the appropriate actions based on their policies.

C-TÆMS

COORDINATORS researchers have jointly defined a common problem domain representation based on the original TÆMS language (Horling *et al.* 1999). The new language, C-TÆMS, provides a semantically sound subset of the original language, representing multi-agent hierarchical tasks with probabilistic expectations on their outcomes (characterized by quality, cost, and duration) and complex hard and soft interactions (Boddy *et al.* 2005). Unlike other hierarchical task representations, C-TÆMS emphasizes complex reasoning about the utility 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 agents and 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. At any time, each C-TÆMS agent can be executing at most one of its methods.

A C-TÆMS model is a discrete stochastic model: methods have multiple possible outcomes. Outcomes dictate the *duration* of the method, its *quality*, and its *cost*. Quality and cost are unit-less, and there is no fixed scheme for combining them into utilities. For the initial COORDINATORS experiments, we dispense with cost, so that quality may be thought of as utility. Methods that violate their temporal constraints yield zero quality (and are said to have *failed*).

Every task in the hierarchy has associated with it a "quality accumulation function" (QAF) that describes how the quality of its children are 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 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 later cannot contribute quality to the parent task.

Traditional planning languages model interactions between agents and the state of their 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. Complicating matters significantly is the fact that these NLEs may have associated delays. We will discuss the implications of all of these in terms of developing Markov models of the COORDINATORS problem shortly.

Figure 1 illustrates a simple version of the two-agent hostage-rescue problem described earlier. 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 agents in a COORDINATORS problem 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).

Execution environment

COORDINATORS agent designs are evaluated in a real-time simulation environment. When agents first wake up in the simulation they are given their local, subjective view problem description and an initial schedule of activities. The initial schedule captures the notion that the agents already have some nominal plan of what they should do. One of the prime objectives of COORDINATORS is to manage responses to uncertain outcomes and task model changes.

During evaluation, the agents send commands to the simulator specifying what methods they want to initiate, and the simulator randomly determines duration and outcome quality according to the distributions in the objective C-TÆMS model. In order to achieve the best results, agents will need to adapt their courses of action to the simulated events. Agents can use a combination of precomputation (policy generation) and reaction to adapt their plans. However, the simulated environment won't wait for the agents to replan; each simulated time tick is tied firmly to wall-clock time.

In addition to the stochastic outcomes that are modeled in C-TÆMS, the simulator can inject unmodeled events. Such unmodeled events can include adding nodes to the C-TÆMS network, removing nodes, or modifying duration/quality distributions for existing nodes. Agents must manage their plans at runtime to account for these changes, and they must also contend with the fact that only a subset of the agents may be aware of the unexpected change.

Multiagent MDPs for distributed plan management

Given a (fixed) C-TÆMS task network and the fact that method outcomes are stochastic, we frame the problem as building a *policy* that dictates how each agent in the network chooses methods at every point in time. In earlier work on TÆMS, the objective was to find a satisfactory balance among some combination of quality, cost, and duration (Wagner, Garvey, & Lesser 1998). In C-TÆMS, by contrast, the problem is simply to find a policy that maximizes the network's expected quality (utility).

Markov Decision Processes

We assume that the reader has a grasp of the basic definitions of Markov Decision Processes; we recommend Puterman's text (Puterman 1994) for more specifics. Briefly, an MDP is akin to a finite state machine, except that transitions are probabilistic, rather than deterministic or nondeterministic. Agents may also receive reward (which may be either positive or negative) for entering some states. Typically, this reward is additive over any trajectory through the state space (some adjustments are needed in the case of MDPs of infinite duration). The solution to an MDP is a *policy* — an assignment of action choice to every state in the MDP — that maximizes *expected utility*. MDPs' advantages are that they offer a sound theoretical basis for decision-making and action under uncertainty, and that there are relatively simple, polynomial algorithms for finding optimal policies.[1]

An agent's C-TÆMS task model may be thought of as specifying a *finite-horizon* MDP. The problems are finite-horizon because C-TÆMS problems have finite duration, with no looping or method retries. However, the MDP tends

[1]But polynomial in the (often large) size of the state space!

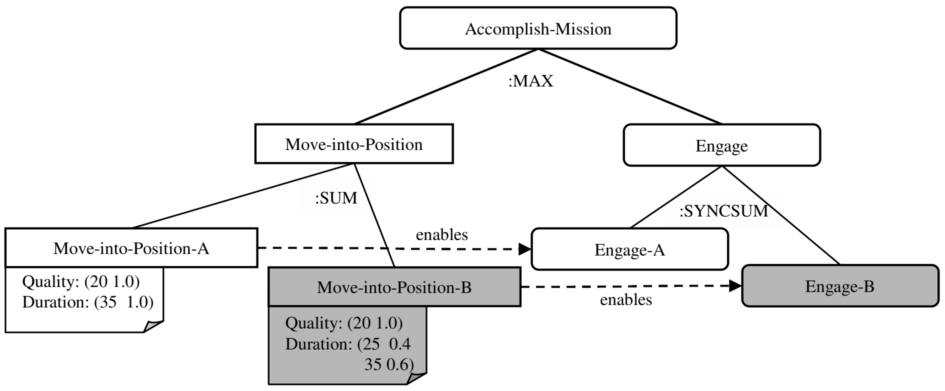

Figure 1: A simple C-TÆMS task network for two agents, illustrating some of the representation features. Some details have been omitted for brevity.

to be quite large for even modest-sized C-TÆMS problems because of the branching factor associated with uncertain outcomes, and because of the temporal component of the problem. Multi-agent C-TÆMS MDPs are even worse.

The COORDINATORS problem differs from most problems treated as MDPs in two important ways. First, the problem is inherently distributed and multi-agent, so that in the objective view, multiple actions can be executed simultaneously. For this reason, if one were to formulate a centralized COORDINATORS problem directly as an MDP, the action space would have to be a tuple of assignments of actions to each agent. As one would expect, this causes an explosion in the state space of the problem. A second difference is that the actions in the COORDINATORS domain are temporally extended, rather than atomic. Such "durative" actions can be accommodated, but only at the cost of a further explosion in the state space. Other aspects of the COORDINATORS problem make its MDP formulation tricky and increase the state space even more. For example, the delays associated with NLEs such as "enables" links require the MDP state to hold some execution outcome history.

Unrolling TÆMS task nets

We translate C-TÆMS task networks by "unrolling" them into MDPs that make explicit the state space implicit in the task net. For any agent, the C-TÆMS task network defines a possible state space and transition function. A C-TÆMS agent's state may be defined as a tuple: $\langle t, M \rangle$, where t is the current time, and M is a vector of method outcomes. If $\mathcal{M}$ is the set of methods a TÆMS agent can execute, we can assign to $\mathcal{M}$ an arbitrary numbering, $1 \ldots n$ for $n = |\mathcal{M}|$. Then M is a set of tuples, $\langle i, \sigma(i), \delta(i), q(i) \rangle$: the index of the method, its start time, duration, and quality. This information is sufficient (but not always all necessary) to give a state space that has the Markov property. The C-TÆMS task network, with its duration and quality distributions, defines a transition function. For example, if an agent executes a method i starting at time t, yielding a duration $\delta(i)$ and a quality $q(i)$, that is a state transition as follows:

$$\langle t, M \rangle \rightarrow \langle t + \delta(i), M \cup \{\langle i, t, \delta(i), q(i) \rangle\} \rangle$$

Our techniques "unroll" the state space for the MDP from its initial state ($\langle 0, \emptyset \rangle$) forward.[2] From the initial state, the algorithms identify every possible method that could be executed, and for each method every possible combination of duration-and-quality outcomes, generating a new state for each of these possible method-duration-quality outcomes. Each state is then further expanded into each possible successor state, and so on. For states where no methods can apply, a "wait-method" is generated that leads to a later state where some non-wait method has been enabled (or the scenario has ended). The unrolling process ends at leaf states whose time index is the end of scenario. The code for performing this unrolling was adapted from previous state-space unrollers developed at SIFT for applications in optimal cockpit task allocation (Miller, Goldman, & Funk 2003).

There are a number of complexities in performing this unrolling. For example, the set of enabled methods is a complex function of the current state, influenced by temporal constraints, NLEs, etc. Our unroller tries to eliminate

[2]Note that we can just as easily start unrolling *in medias res*, by starting from a state in which the agent has already executed some actions.

dominated action choices without enumerating the resulting states. For example, once one task under a :MIN QAF has failed (gotten zero quality) then none of the :MIN node's remaining children should ever be tried, because they cannot increase the :MIN node's quality.[3]

Furthermore, we can exploit the structure of the C-TÆMS task model, and its QAFs, to collapse together states that are equivalent with respect to future action choice and final quality. These techniques, and especially the equivalent-state folding, give exponential reductions in state space size.

Informed Unrolling

Since full enumeration of even single-agent C-TÆMS MDPs is impractical (the state space is too large), we have developed a technique for heuristic enumeration of a subspace of the full MDP. Our *informed unroller* (IU) algorithm prioritizes the queue 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. The intent is to guide the unrolling algorithm to explore the most-probable states first. Because the probability of reaching a state is dependent on the policy, the IU intersperses policy-formulation (using the Bellman backup algorithm) with unrolling. However, the quality of a state at the edge of the partially-unrolled state space is generally difficult to assess, since quality often only accrues at the end of the entire execution trace (i.e., the domains include delayed reward). Thus, we have developed a suite of alternative heuristics for estimating intermediate state quality.

We have experimented with several different heuristic functions to evaluate edge states and guide the informed unroller. Currently the most consistent performance is achieved by a heuristic that emphasizes unrolling the highest (estimated) probability states first. However, computing the heuristic function and sorting the openlist is an expensive operation. Therefore we constrain the openlist sorting activity in two ways. First, the sorting only occurs when the size of the openlist doubles,[4] and second, once the sorting function takes more than a certain maximum allocated time (e.g., one second), it is never performed again. 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 will refine its model on the fly).

[3]Actually, if a child node can enable some other local or non-local method it may still have utility. This sort of effect makes it quite challenging to accurately assess dominance.

[4]This threshold could, of course, be tailored.

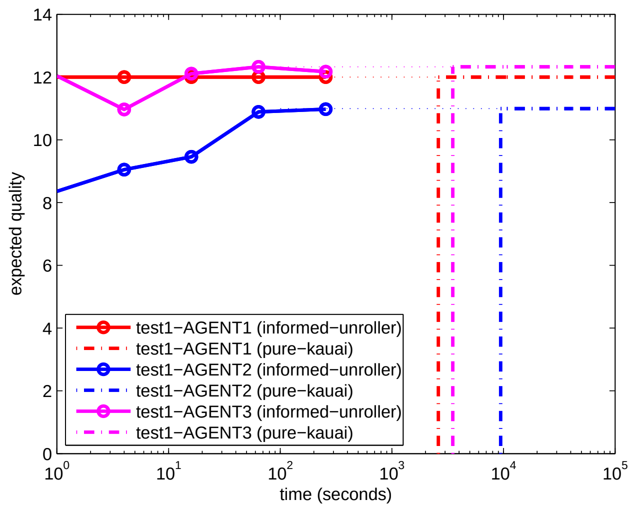

Figure 2: The Informed Unroller can find near-optimal policies much faster than building the complete MDP.

The informed unroller work is at an early stage, but early results from the evaluation against a complete solution 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 return increasingly effective policies given more time. This allows the IU-agent to flexibly trade off the quality and timeliness of its policies. The current version of the IU does not support repeated, incremental unrolling of the state space during execution. However, we are actively working to build a new version, and integrate it into our COORDINATORS agent.

Related Techniques

The IU approach is related to the "approximate dynamic programming" algorithms discussed in the control theory and operations research literature (Bertsekas 2005). These approaches derive approximate solutions to MDP-type problems by estimating, in various ways, the "cost to go" in leaf nodes of a limited-horizon portion of the full state space. While our exploration of the literature is not yet complete, initially we believe that a key difference in our IU approach is the notion of time-dependent horizon control and unrolling-guidance (vs. just estimation of leaf-node reward for policy derivation).

The IU method is also somewhat similar to LAO* (Hansen & Zilberstein 2001), which uses knowledge of the initial state(s) and heuristics to generate a state subspace from which the optimal policy can be provably derived. Our technique differs in substantial ways. The IU 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, 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. LAO* trims the state space by using an admissible heuristic. No such heuristic is available to the IU because estimating the final quality of a C-TÆMS network, given the various QAFs, is so difficult. However, because the IU is an online algorithm, it can be re-employed periodically during execution, so the policy created can be iteratively tailored to better fit the circumstances.

Coordination

When we consider multiple COORDINATOR agents, the problem expands to finding an optimal *joint policy*. This problem is challenging because:

- The number of possible local policies for agents is in general very large, so the product space of joint policies to search through can be astronomical.
- The size and distribution of the problem makes reasoning about the global behavior of the system impossible.

To address these practical limitations on solving the problem of finding optimal joint policies, our coordination approach is designed to take advantage of three assumptions:

- Inter-agent effects are well-defined and visible to individual agents through their subjective views.
- The agents are given reasonable *initial* schedules.
- The sum of the local expected qualities of the agents is a sufficient approximation of the global expected quality for the problem with respect to guiding search toward improved joint policies.

Our coordination method exploits the first assumption by mining an agent's local subjective view to detect the possible "coordination opportunities." For example, if one of agent-A's methods enables a method for agent-B, that represents a coordination opportunity. Given these discrete opportunities, we can reduce the overall problem from an enormous search over the space of joint policies to a merely huge search over the space of alternative commitments on coordination opportunities.

We exploit the second assumption by extracting default initial coordination decisions from the nominal initial schedule of activities. After ensuring that the agents have coherent and consistent expectations about those commitments, they can then search for approximately-optimal local policies where the commitments for their coordination opportunities are enforced, as described later.

The third assumption will enable negotiating agents to compare different sets of commitments. For an individual agent, a commitment set is preferred to another if it enables the creation of a local policy that has a higher expected reward. A group of agents will prefer a particular combination of local commitment sets if the sum of the expected qualities of their resulting local policies is higher. Note that some agents in the group might have lower expected qualities, but if these are more than compensated by the expected quality gains of other agents then the combination of local commitment sets is considered superior.

There are several ways in which these assumptions and our solution approach (coordination over commitments) may result in sub-optimal behavior. For example, the actual optimal policy set may not adhere to a static set of commitments: e.g., to behave optimally, agents may have to adjust which enablements they will accomplish depending on how prior methods execute. To mitigate this weakness, we plan to have our agents deliberating and negotiating continually, so that they can manage and adapt their commitment set and policies on the fly as methods execute.

Perhaps worse, the third assumption may be violated: an agent's subjective view may not give an accurate estimate of global quality. This problem arises because of the non-monotonic and non-additive effects of different QAFs. For example, suppose three agents have methods under a common :SYNCSUM parent, and in the initial schedule, two of them have agreed to execute their methods at time 10, while the third agent has decided not to execute his method. Now, suppose the third agent realizes it can execute its method at time 7. If it chooses to do so, its local C-TÆMS model may indicate that this change can produce increased local quality and therefore looks like a good idea. But if the other agents do not adjust their plans, then their methods running at time 10 will be rendered useless (failed, due to the :SYNCSUM semantics emulating the lack of surprise). Thus the third agent may inadvertently reduce the overall team quality by pursuing its local goals without proper consideration of global effects in the task model. This example is fairly easy to recognize, but in general such effects can propagate across multiple NLE links, making them very hard to recognize and manage. Further study is required to determine how badly this assumption affects the performance of our COORDINATOR agent societies in widely varied problems.

Initial Commitments

The agents begin their coordination by identifying their potential interactions (coordination opportunities) and initializing their planned responses (commitments) to those interactions. Each agent does this based on its subjective view of the problem and on its corresponding initial schedule.

The subjective view includes information on the agent's own tasks, as well as some limited information on the tasks of other agents with which the agent may need to coordinate. Specifically, the agent will know about the existence, though not the details, of other agents' tasks that are connected to local tasks through *non-local effects*, for example tasks that another agent must execute before some local task can start. Several types of *coordination opportunities* are extracted from the agent's subjective view:

- *NLE* coordination opportunities are indicated by NLEs across multiple agents' tasks.
- *Synchronization* coordination opportunities are indicated by :SYNCSUM tasks in the subjective view.
- *Redundant Task* coordination opportunities are indicated by tasks that are visible to other agents, indicating that they too have children under that node. Depending on the QAF, such shared tasks may indicate that only one agent should execute methods below the task.

Initial Commitments

The initial schedule is used to infer a set of provisional commitments to the coordination opportunities. For example, if agent-A has a method M_A that enables agent-B's method M_B, then both agents recognize an NLE coordination opportunity. Each agent inspects its initial schedule to see which methods are initially scheduled. For example, agent-A may find that its initial schedule suggests executing M_A at time 3, with an anticipated finish at time 9, and agent-B's initial schedule may suggest executing M_B at time 11. Agent-A would form a tentative commitment to finish M_A before 9, and agent-B would form a tentative commitment expecting its enablement before time 11. Note that the agents' initial schedules may have flaws or poorly-aligned methods, so these initial commitments may not be entirely consistent.

Distributed Constraint Satisfaction

In our problem formulation, negotiation can be viewed as a search over the space of compatible commitments that agents make to their coordination opportunities, seeking a set of commitments that satisfactorily achieves high global expected quality (as estimated by the sum of local expected qualities). The search begins with the initial set of commitments, and then tries to improve on this initial set (or some previous improvement of that set) to increase the compatibility and/or quality of the commitments.

Our agents begin by ensuring compatibility among the commitments, essentially treating the problem as a distributed constraint satisfaction problem (DCSP). For example, an agent acting as the source for an enablement NLE should be committed to completing the enabling task before the target agent expects to begin the enabled task. If their locally-generated commitments do not satisfy this constraint, then the agents need to resolve the inconsistency.

As a simple first step, our agents exchange their initial commitments with the relevant partner agents and, in a one-shot computation, modify the commitments to ensure consistency. Since each agent applies the same process to the same information to establish consistency, each agent involved in a coordination opportunity will arrive at the same resolution to any inconsistency. Note that, even within this simple protocol, there are several possible approaches. For example, if agents disagree on the time an enablement will occur, the agreed-upon time could be the earlier time, later time, or some intermediate value.

Enforcing Commitments

Given a consistent set of commitments, we need a method to *enforce* those commitments in the agents' policy-finding process. One approach would be to re-write the task model so that the MDP unroller does not even consider the possibility of violating the commitment. This approach is too restrictive because it would not permit the agents to reason flexibly about off-nominal method outcomes. In some situations it may be a rational decision to *not* satisfy a given commitment. Therefore, rather than rigidly enforce commitments, we strongly bias the policies generated by the MDP solver towards the committed behaviors. This can be done by providing an extra reward for states in which a given commitment is satisfied and an extra penalty for states in which it can be determined that the commitment will not be satisfied. We also add local "proxy methods" modeling the execution of other agents' tasks.

We simplify the negotiation process to reduce the complexity of enforcement. For example, the execution of tasks by other agents is currently modeled deterministically: the commitment agreement includes a single time by which the task is to be executed. A more expressive representation would accommodate completion time distributions.

Conclusion and Future Directions

Our agents are in the early stages of their design evolution, having "played the game" in simulation for only a short while, during which time the simulation environment itself has also evolved. In their current form, the agents recognize coordination opportunities, derive initial commitments from the initial schedule, communicate to ensure consistency among commitments, and then unroll the resulting local MDP until their first method's release time arrives. They then execute their MDP policy whenever possible. If the MDP unrolling expands the entire reachable state space, they should remain "on-policy" and perform according to the MDP's predictions unless there is an unmodeled failure or new task arrival. More often, the unrolling is incomplete and the agents can "fall off-policy" when the simulated world state reaches an unexpanded part of the MDP state space. At that point, the agents begin executing one of two simple reactive scheduling algorithms (one driven off the initial schedule, one based on a one-step lookahead).

Initial performance results indicate that, given a modest amount of time to unroll their MDPs (e.g., twenty seconds), the agents can explore tens of thousands of MDP states and sometimes dramatically outperform the simple reactive schemes alone. However, there is clearly room for major improvement, and the simple reactive approaches likewise sometimes dramatically outperform our initial IU system. As our system evolves to fulfill its full design, we will conduct more detailed experimentation and analysis.

We believe that we now have a strong foundation for principled future work on coordinated plan and schedule management in uncertain domains. We have many additional features to develop, including incremental and continuous MDP unrolling (so the agents continue to think about the problem as they are executing their existing policy) and improved negotiation that will lead to more dramatic changes in the initial schedule and inter-agent commitments.

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

Bertsekas, D. P. 2005. Dynamic programming and suboptimal control: A survey from ADP to MPC. In *Proc. Conf. on Decision and Control*.

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

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

Horling, B.; Lesser, V.; Vincent, R.; Wagner, T.; Raja, A.; Zhang, S.; Decker, K.; and Garvey, A. 1999. The TAEMS white paper. Technical report, University of Massachussetts, Amherst, Computer Science Department.

Miller, C. A.; Goldman, R. P.; and Funk, H. B. 2003. A Markov decision process approach to human/machine function allocation in optionally piloted vehicles. In *Proceedings of FORUM 59, the Annual Meeting of the American Helicopter Society*.

Puterman, M. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons.

Wagner, T. A.; Garvey, A. J.; and Lesser, V. R. 1998. Criteria Directed Task Scheduling. *Journal for Approximate Reasoning* 19:91–118.