Coordinating Highly Contingent Plans: Biasing Distributed MDPs Towards Cooperative Behavior

Robert P. Goldman and **David J. Musliner**
SIFT, LLC
{rpgoldman,musliner}@sift.info

Edmund H. Durfee
University of Michigan
durfee@umich.edu

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

Introduction

In this paper, we describe a "top down" technique for finding approximately optimal joint policies for groups of cooperating agents in stochastic environments. Our technique is distinguished from previous methods for such problems in that the agents' policies are driven by commitments to each other, rather than having coordination arise bottom-up, driven by features of the policies of individuals.

Optimal coordination in stochastic multiagent environments is notoriously difficult because of the combinatorial number of joint states and joint actions, which renders the problem in the general case NEXP-complete (Goldman & Zilberstein 2004). Additional challenges are imposed by the inability for distributed agents to maintain full awareness of system progress through the joint state space. As a consequence, approaches to solving this problem either sacrifice generality (by, for example, assuming particular forms of independence between agents or full observability of the global state by all agents (e.g., (Becker *et al.* 2004))), or sacrifice optimality (by, for example, finding only locally-optimal solutions (Nair *et al.* 2003)). Our work must make its own concessions to computability, but makes different choices.

Consider the very simple cooperative multiagent problem shown in Figure 1. In this example, Agent A can either perform Method-A1 or Method-A2, where the former has an expected reward of 0.8 (an 80% chance of achieving 1 unit, and 20% chance of achieving zero), and the latter has an expected reward of 100. From a purely local perspective, the choice for Agent A is obvious: it should execute Method-A2. Meanwhile, Agent B can either perform Method-B1 or Method-B2. Method-B1 has an expected reward of 8000 (80% probability of a reward of 10000) if Agent A has previously executed Method-A1. But it gets zero reward if Agent A does not execute Method-A1. Alternatively, Method-B2 has an expected reward of 100 regardless of Agent A's action. As a team, the agents get the sum of the local rewards that each accrues.

A key aspect of this example is that the agents face a tension between taking actions that depend on each others' activities, versus taking actions that achieve local reward unilaterally. The agents are not independent of each other, and so each should formulate a policy based on expectations of the policies (or the effects of policies) of the agents that could influence it. This idea has been studied in the literature in work such as that of Nair, in which agents iteratively formulate local policies and then exchange these. Knowing the policies that other agents are planning on following, an agent can use those policies to update local transition probability models, and thereby derive an adjusted local policy that takes into account the likely behaviors of the others. Of course, other agents are adjusting their policies as well, so this process may require multiple rounds of iteration for the policies to converge.

The approach we present in this paper uses a similar idea of having each of the agents formulate its local policy that will be part of a joint policy, but does not follow the bottom-up process of searching through a portion of the joint policy space by iteratively forming and exchanging local policies until convergence. Our "top-down" strategy assumes that agents are aware of the ways that they can influence each other. In particular, as shown by our example, agents with explicit models of their task hierarchies and task dependencies know about opportunities for coordinated behavior from the outset. Moreover, we assume that these interactions are sparse compared to relationships between the tasks local to an agent.Rather than iteratively formulating complex local policies that get passed between the agents to incrementally tweak the policies of others, our approach involves having agents first reach agreement on *commitments* about their interactions. These commitments are intended to be *categorical* promises to achieve some objective or to perform some task. The agents use those commitments to bias their local policy formulation processes such that each agent builds a policy that strives to meet the commitments to which it has agreed, and also account for failures.

Unrolling Task Models into MDPs

Our approach to contingent planning starts with a *hierarchical task network* (HTN) model like the one presented in Figure 1. These task models provide very concise, expressive ways of specifying multi-agent planning problems. We would like to apply powerful, generic algorithms for Markov Decision Problems (MDPs) to the sequential decision problems (regarding which tasks to execute when) that are implicit in the task models. MDP algorithms typically operate on the state spaces of MDPs, represented more or less explicitly. We have developed algorithms that "unroll" the decision problem for each agent defined in our task models into a MDP, capturing alternative choices about which task to pursue in a particular state and the possible states resulting from executing a chosen task. The agents then construct a joint policy implicitly; each agent computing its own local policy. The agents' local policy generation uses inter-agent *commitments* to bias local policy generation to improve cooperation among the agents.

Task Models

Our approach to contingent multiagent planning starts with a *hierarchical task network* (Nau, Ghallab, & Traverso 2004) like the one presented in Figure 1. The actual task model format that our system uses as its input is the C-TÆMS dialect (Boddy *et al.* 2005) of the TÆMS task modeling language (Horling *et al.* 1999), but the particular details of TÆMS are not critical to our discussion here; we will be focusing on aspects that are common to all flavors of HTNs. In particular, our HTNs will contain *tasks* that may be performed by the various agents, and top level tasks are performed by performing special subsets of their subtasks (child nodes); we say that tasks may be *decomposed* into subtasks. For example, in Figure 1, "Agent-A tasks" may be decomposed into "Method-A1" or "Method-A2," or both. One oddity of TÆMS terminology that will affect the reader is that in TÆMS internal nodes are referred to as *tasks* and leaf nodes as *methods*; this differs from standard HTN terminology.

There are two features of TÆMS that are critical to our discussion here. The first is that TÆMS problems are optimization problems. Groups of agents receive reward (referred to as "quality") for performing sets of tasks. Note that the reward is incurred by the collective as a whole; agents are not individually self-interested. The second important feature is that tasks can interact with each other. In particular, as in Figure 1, tasks can *enable* each other. In this example, no quality will be accrued by Agent-B from performing Method-B2 unless Agent-A has already successfully completed Method-A1. These are the features that make TÆMS particularly suited to *multiagent* task modeling.

TÆMS task models have temporal aspects. TÆMS methods are temporally extended, rather than instantaneous, and there may be uncertainty about the duration of methods. Furthermore, TÆMS tasks may have temporal constraints: deadlines and release times (activities may not start before their release times).

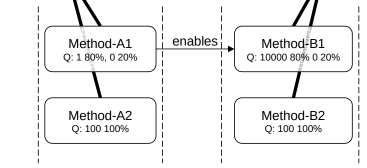

Figure 1: This simple multi-agent additive task model shows time-constrained, uncertain-outcome tasks for two agents.

MDPs

We do not have space here for a thorough introduction to 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 and agents may receive reward for entering certain states. Typically, this reward is additive over any trajectory through the state space (some adjustments are needed in the case of MDPs of unbounded duration).

More formally, an MDP may be defined as a tuple: $\langle \Omega, \omega_0, A, T, R \rangle$ made up of a state space, Ω, an initial state, ω_0, a set of actions, A, that the agent may perform, a transition function, $T : \Omega \times A \mapsto P \times A$, and a reward function, $R : \Omega \mapsto \Re$. The transition function defines a probability distribution over successor states, conditioned on taking some action in a previous state, and the reward function defines a reward conditional on reaching a particular state.MDPs may be either *finite*- or *infinite*- horizon problems; the problems that we are concerned with here are finite-horizon problems.The solution to an MDP is a *policy*, an assignment of action choice to every state in the MDP.Typically, one searches for a policy that is optimal in the sense of maximizing *expected reward*.

Unrolling Task Models

For any agent, the task network implicitly defines a possible state space and a transition function that maps individual states to possible future states (much of the rest of this paper addresses the decomposition of the overall problem into a set of single-agent MDPs). Our system reasons about task networks by "unrolling" them into MDPs that make this implicit state space explicit. In this section, we describe the state space, explain how the transition function is defined and describe our unrolling algorithm.

State space. In order to satisfy the Markov property, we must incorporate in each state enough information to make the state transition function be a function of only the current state and the chosen action. We must also incorporate into each state enough information that we can evaluate the reward function for the task network. Doing so is complicated somewhat by the temporal aspects of the problem.

There are two primary methods for representing temporal problems as MDPs. The first is to have a "clocked" MDP where each state represents the state at an instant of time, so for a trajectory of ten time units, there would be ten states (and nine transitions). This kind of representation works well when the actions have relatively short and regular durations. The alternative is to make the time value part of the state of the MDP and have states be temporally extended. So, for example, if the agent chooses to perform a method of duration ten, then there would be a transition from a state with $t = 1$ to one with $t = 11$. For the task models with which we are working, having an explicit time component to the state works better; the methods have widely-varying durations, and the clocked models are too large.

To compute the reward for our task networks, we need to record the completion time and quality of each executed method. We must do this because inter-task relationships such as enablement mean that we cannot immediately determine all of the effects of performing a given action.

Because methods have duration and release time constraints, we must record "wall-clock time" in the state of the agent. That is, in order to know the possible outcomes of executing a method (recall that the outcome is characterized by duration and quality), we must know when the action is starting. When determining the outcome distribution for a method M, we must also know the quality of every node n for which there exists an NLE, $n \rightarrow M$, and we must know the quality of n at the time $now - \mathrm{delay}(n \rightarrow M)$.[1] The need to reason about NLEs is another reason we must record the completion times of all methods. Because of the delays in the NLE effects, we must be able to compute a quality for the tail nodes at different times in the trace.[2]

Given the above features of C-TÆMS, we can define the state of a C-TÆMS agent as a tuple $\langle t, M \rangle$, where t is the current time, and M is a set 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 $1...n$, 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.[3] This information is sufficient (but not always all necessary) to give a state space that has the Markov property.

Transition function. The 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:

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

In addition to allowing agents to carry out C-TÆMS methods, we also allow them to execute "wait" pseudo-actions, but we will not discuss those further here. Recall that we do *not* generate states at every tick in the system clock. Instead, we only generate states for "interesting" times, at which methods start or finish.

Unrolling the state space. Our techniques "unroll" the state space for the MDP from its initial state ($\langle 0, \emptyset \rangle$) forward.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).

Even with aggressive pruning, we are unable to fully enumerate the state spaces of even a single agent's MDP. Instead, we must use heuristic search to generate a subproblem; we discuss this elsewhere (for example, see (Wu & Durfee 2007)). For this paper, it is sufficient to know that the state spaces of the individual agents are prohibitively large; we will see that this leads us to use methods that serve to reduce the state space while coordinating the activities of multiple agents.

[1]This is actually a slight oversimplification. For some NLEs we need to know the exact quality of the tail node, for others we need only know whether that quality is non-zero.

[2]Note that the quality of nodes is a montonically non-decreasing function of time.

[3]In practice, the set M is most efficiently implemented as a vector.

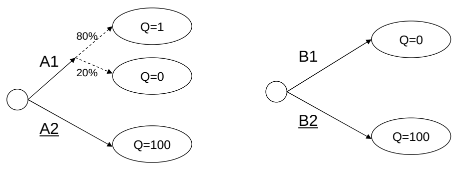

Figure 2: Without coordination, the agents build locally-optimal policies that do not coordinate well, and achieve poor results overall for the team.

Example in Detail

We now return to the example in Figure 1 for a more detailed analysis. In this highly-simplified problem, each agent has one decision with three possible choices: execute Method-A1 (resp. -B1), Method-A2 (B2), or no method. This results in nine possible joint executions, some with uncertain outcomes.

This is not quite the same as saying that each agent has three possible local *policies*, resulting in nine joint policies. Roughly speaking, the number of possible joint policies is exponential in the number of agents, with a base that is the cross-product of action choices and locally-discernable states for each agent. As the agents are granted more information about each other, and as the length of decision sequences grows, that base will grow extremely quickly.

One way to reduce the combinatorics of the problem is to generate a policy for each agent based only on local information, combining these local policies into a global policy.

In the example in Figure 1, if each agent formulates its local policy, Agent A will quickly converge on doing A2, maximizing its local expected utility. Agent B, with no information regarding whether Agent A will execute A1, assumes not and similarly decides to execute B2, resulting in a joint policy as shown in Figure 2, achieving a total reward of 200. If, however, Agent A and Agent B have some way to agree that Agent A will execute A1, thus enabling B1, then the optimal joint policy (executing A1 and B1) results in a 64% chance of a reward of 10001, a 16% chance of a reward of 1, and a 20% chance of a reward of 0, for a total expected reward of 6400.8.

If the agents have some capability for communicating at run-time, they can do better still. Despite agreeing to the enablement, Agent A may try and fail to achieve it. If Agent B can know the outcome of Agent A's attempt at A1 *before* it begins executing its own method, then Agent B's policy can branch on the uncertain outcomes of A1. If A1 fails, then Agent B should execute B2, otherwise B1. This improved policy raises the agents' expected reward from 6400.8 to 6420.8, and ensures that the team will always get a reward.

Biasing Policy Generation Using Commitments

In this section, we describe the implemented mechanisms whereby agents are "encouraged" to construct local policies that favor cooperative behavior. The enforcement mechanism that we use to guide policy generation in this way has two parts, one for the agent using a commitment, and one for the committed agent.

Consuming commitments.

An agent that has received a commitment from another agent should build a policy that exploits that commitment. In our example, `B` has a commitment from `A` that A will execute `A1` to enable B's `A2` by time `Time-1`. We need B to build an MDP that recognizes that `A2` is not enabled before `Time-1`, but is expected to be enabled afterwards. One simple approach would be to unroll the MDP from a modified model in which the target method is not considered executable until the enablement is expected.

However, this precludes any explicit representation of the fact that A may not succeed with the enablement, either because it chooses not to execute `A1` or because `A1` does not achieve any quality. So, we need a means to represent within the local MDP the possibility that the enablement might fail. To accomplish this, the MDP-generating code builds "proxy methods" that correspond to other agents' commitments.

This ensures that the agent consuming the commitment will find the appropriate methods enabled at the appropriate time. When unrolling the MDP out of the task model description, the unroller inserts the expected quality for the proxy methods at the appropriate times. The result is that the local MDP policy will take advantage of the expected inter-agent enablement commitment if it is the "right thing to do"— that is, if it maximizes the local expected quality. As an immediate consequence, if A fails in the enablement and B is notified, B should instead choose to execute `B2` during that time slot. Figure 3 shows the optimal policies for the model including a proxy method.

Keeping commitments.

The commitments made *by* an agent are handled by a different mechanism, akin to that used by model-checking systems for program verification. We may describe the commitments made by our agents in terms of temporal logic formulas. In particular, for enablement commitments, an agent is promising to complete a method before a certain time. Such a proposition may readily be checked by walking over the state space of the MDP — at any state this assertion is either satisfied, violated, or is still pending. Note that no additional traversal of the state space is necessary: the checking can be done as the task model is unrolled into an MDP state space. Should the agent satisfy a commitment, it receives a reward to its local MDP and if the agent violates a commitment, it is penalized.

One problematic issue is how to assign reward (resp., penalty) to commitment satisfaction (violation). It is not sufficient to simply impose a penalty on a commitment failure proportional to the payoff of the method whose enablement has failed. The problem with this approach is that the follow-on effects can be arbitrarily bad, because of chains of enablement. Worse, the agents focus their policy-finding efforts on parts of the state space in which the commitments are kept. If too many commitments are violated, or if a single critical commitment is violated, then agents may find themselves in a part of the state space for which they have no policy computed. Accordingly, we heavily reward (penalize) the agents for fulfilling (violating) their commitments.

Generalizing Commitments

For the purposes of clarity and brevity, the discussion in this paper is limited to commitments involving actions enabling other actions. The implemented system described in (Musliner *et al.* 2007) includes support for commitments between agents covering a variety of other relations between actions, including *disablement* ("don't do your action until after I start my action"), *facilitation*, and *hindering*. Commitments are also used to allocate responsibility for *shared tasks*.

More Complicated Example

Figure 4 illustrates a modified version of our earlier example domain, where the `A` tasks are identical but `B` has two new methods which run in the earlier time window, with time available to run only one of them. The new `B3` can be used to enable `B2`, or `B4` can be used to achieve higher local quality but not enable `B2`. The interesting feature of this domain is that `B` must decide whether to execute `B3` or `B4`, before finding out whether `A1` has been executed and generated positive quality, thus enabling `B1`.

Figure 5 shows the globally-optimal policy for `B` for this example (`A`'s policy is unchanged: execute `A1`). The underlined labels represent action choices appearing in the optimal policy. Note that some of the policy branches depend on the "outcome" of a proxy method for `A1`.

Choosing Commitments

In this section, we turn to the question of how agents can choose good commitments, and how they can reach agreement on a consistent set of such commitments. We continue to assume that global optimality is unrealistic computationally, because it can require an exhaustive search over the space of all possible combinations of commitments for all

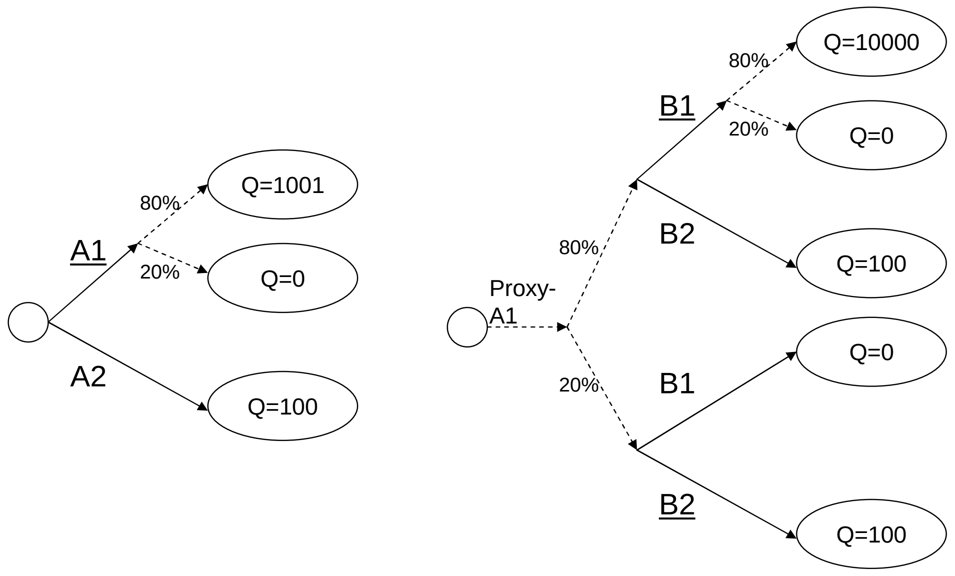

Figure 3: With proxy methods and commitment rewards biasing the MDPs towards the commitment, the agents build and execute these (optimal) policies.

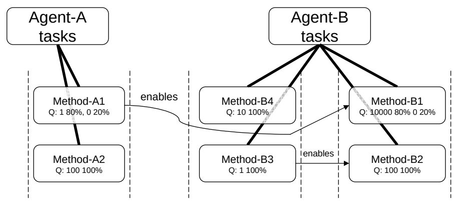

Figure 4: A task model with a more complex tradeoff.

potential agent interactions. We thus focus on developing local search techniques for converging on "good enough" commitments.

In the rest of this section, we present the different pieces of the commitment negotiation process. First, we show how agents identify the space of possible opportunities for coordination. Then, we discuss the process by which the agents can converge on consistent agreements, cast as a distributed constraint local optimization process. Finally, we describe how this process can be used as part of an iterative, hill-climbing local search for agreements over combinations of commitments.

Schematically, the process is as follows:

- Identify opportunities for coordination
- Initialize tentative commitments
- Perform distributed hill-climbing search
  - exchange commitments
  - make local and received commitments consistent
  - build local (quick-and-dirty) policy given consistent commitments

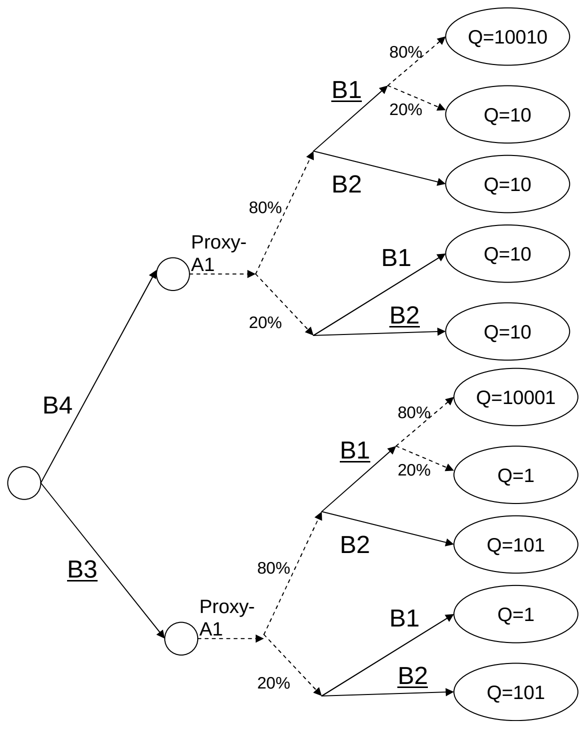

Figure 5: B's coordinated MDP policy for the domain in Figure 4 shows how it trades off near-term local reward to instead enable the potential reward available if the coordination fails.

- – extract new commitments from policies
- – loop back to passing commitments around
- – terminate when no improvement, or timeout

Coordination Opportunities

Each agent begins to reason about commitments by first identifying the possible interactions it can have with other agents, and treating each of these as a *coordination opportunity*. To identify coordination opportunities, each agent analyzes its task model to find incoming and outgoing relationships such as enablement. We assume that this information is captured explicitly in the task model, if it is not it can be derived through an exchange of information between agents about the preconditions and effects of their various actions (Clement, Durfee, & Barrett 2007).

A single coordination opportunity such as an enablement can potentially involve an arbitrary number of agents depending on where in the task model the enablement occurs. For example, in Figure 1, had the top-level "Team Task" itself been the source of an enablement (if this team of A and B were working together to enable some other team of agents), then both A and B would model the coordination opportunity.

Agents may also interact because of how their local tasks combine into collective reward. For example, in Figure 1, the agents interact by both contributing to the top-level task, which sums their indvidual qualities. Had the top-level task instead accrued quality by taking the *minimum* quality of Agent A and Agent B, then a better coordinated response by the agents would have been to each take their local actions.

Initial Commitments

Commitment negotiations must be an "anytime" process, because the amount of time available to negotiate might fall short of what would be needed to systematically ensure an optimal solution. Our approach is to conduct a local search, beginning with a set of tentative commitments, and then to iteratively improve those commitments until they cannot be improved further, or until time runs out, at which point we use the current best set of commitments.

This approach requires an initial set of commitments for the coordination opportunities as a starting point. Each of our agents analyzes the possible state or states that it expects to reach once all of its tasks have been performed. Because the state representation captures the history of what methods the agent executed, at what times, and with what outcomes, the agent can look at such a state to determine which coordination opportunities would have had to have been successfully achieved to have reached that state. We say that the agent extracts the implied commitments from the state. It can then initialize its commitments to the values of these implied commitments, where coordination opportunities that were not implied to be satisfied can be initialized to null-commitments.

This begs the question of where these final states would come from. One possible source is a nominal schedule of activities that the agent is planning to pursue. This pre-existing schedule of intended activities can be used by the agent to project out to the state that it would achieve, and commitments can be extracted from that state. If the agents in the multiagent system had coordinated these pre-existing schedules (as in the COORDINATORS scenarios), then the commitments that each extracts from its local schedule are very likely compatible with the commitments extracted by others.

Another source for initial commitments uses the same idea as basing them off of a pre-existing schedule, but instead uses a pre-existing or proposed policy. Because a policy represents alternative possible branches, it does not lead to a specific expected final state. Thus, an agent that is extracting commitments from a policy can forward-simulate through the policy probabilistically to find a set of possible final states and a probability distribution over these states. Utilizing the same techniques as before, the agent can extract commitments from these alternative states, and use the probabilities to assess the probability that a commitment will be achieved by the policy. By using suitable probability thresholds, the agent can determine which commitments are likely enough to occur, and can explicitly adopt these as its initial commitments.

Commitment Consistency

Inconsistent beliefs about inter-agent commitments generally lead to poorly coordinated behavior and lower collective reward. For example, in Figure 1 (if we ignore do-nothing policies) there are four possible joint policies. Two are coordinated: either the two agents agree that A will enable B, or they agree that the enablement will not happen. The other two combinations are not coordinated, and both lead to worse expected reward than either of the coordinated combinations. In one combination, A is enabling B, but B is doing its local method, wasting A's effort. In the other combination, A is doing its local method, but B executes (and fails) its enablement-requiring method because it is expecting to be enabled.

We achieve consistent sets of commitments using procedures that take as input a set of tentative commitments to a particular coordination opportunity, and return a modified set of commitments that is assured to be consistent. For example, in the case of an enablement coordination opportunity, if the agent that is the source of the enablement has proposed a null-commitment while the target has proposed a positive commitment to the enablement, the procedure turns both commitments into null-commitments (because if the source will not make the commitment, the target better not assume it). On the other hand, if the source proposes a commitment while the target did not assume it would be enabled, then the procedure converges on having both agents assume the commitment (because if the source is doing the enablement anyway, the target might as well model it to see if it could benefit). Finally, even if both agents assume a positive commitment for the enablement, they might disagree on when the source will be assured of finishing the enabling task and when the target can safely begin the enabled task. The procedure then assigns a heuristically-chosen consistent time for both the source and target commitments.

Agents following this procedure are assured to resolve inconsistencies in the same way. Thus, if each agent sends its initial commitments to other agents and the agents all apply these procedures, they will all find the same set of consistent commitment agreements.

Commitment Improvement

The process described above ensures consistency in the agents' commitments. But it does not ensure that those commitments are as good as they could be, or even feasible. By feasible, we mean that agents might reach a set of commitment agreements that turn out to be impossible for one or more agents to fulfill. For example, in Figure 1, suppose that A2 also provided an enablement for a task of some agent C. One possible set of commitments could involve A committing to enable B and also C, where B and C each expect to be enabled. Of course, while these agreements are individually consistent, they are collectively infeasible.

Agents should attempt to improve their commitment agreements to at least achieve feasibility. To do this we use an iterative, hill-climbing process based on some of the same intuitions as Nair *et al.* (2003)'s approach to bottom-up convergence on joint policies, but avoiding the heavy computational load of his approach because our agents coordinate at the level of commitments rather than exchanging policies.

It turns out that a reasonable set of commitments can be extracted using the (vastly simpler) solution to an MDP constructed using deterministic approximations to actions with uncertain outcomes. The policy generated from this "quick and dirty" model is then used to generate a probabilistic distribution of final states, extracting a set of initial commitments from that distribution. This allows an agent to identify not only commitments that are not likely to be kept in a full policy, but also commitments that might not have been promised but that turn out to be feasible.

The agents iteratively improve their commitments using a distributed constraint optimization protocol. After each agent has formed consistent agreements about its commitments, it uses the quick-and-dirty model, biasing that model based on the initial commitments, to generate a quick-and-dirty policy. It then extracts the commitments implied by this policy. It compares these commitments with the commitments that it used in building the policy, counting the number of differences between the two commitment sets.

The agents exchange their new commitments, and broadcast their counts of differences. If the total number of differences across all agents is zero, then the agents have converged on a set of commitments that no agent wants to change, and so the process terminates. Otherwise, the agents compare this total count to the total count from the prior round of this iterative protocol. If the total has not decreased for some user-specified number of iterations, then the agents have reached a local optimum, and the protocol terminates. If the total count has decreased, the agents repeat the process, generating consistent agreements and then individually checking these with their quick-and-dirty models as before.

Once this iterative process terminates or times out, the agents adopt the final sets of commitments as their negotiated agreements.

Anytime Commitment Search

The distributed hill-climbing process for commitment improvement described above can itself serve as the inner loop of a more thorough search through the commitment space. Hill-climbing search techniques can be improved by introducing the notion of random restart, restarting the hill-climbing process from a different initial state, in the hopes of finding a different local maximum This approach works with the negotiation process described above, as well.

The local maximum in one iteration is compared with the best local maximum from previous local searches, and the best of these commitment sets is saved. This comparison must be made globally the agents collectively need to decide which set of commitments is better, since if different agents adopt different commitment sets we suffer from the problems of inconsistency all over again.

This process of reinitializing the commitments and then hill-climbing from there can be repeated as often as time permits. Our implementation supports this iterative-restart hill-climbing process, though for most of our experiments in the COORDINATORs application domain we turned this off, because in that problem domain our agents always began with initial schedules, so the initial commitment values tended TO start the hill-climbing in a good place.

Experiments

We conclude this section with an analysis of how the techniques we have just presented perform on simple problems where the "right" answers can be identified, to allow comparison. These techniques have been extensively used in COORDINATORs problems involving dozens of agents and hundreds of tasks, scaling well in both handling large numbers of agents and complex local state spaces.

All of the example problems described in this section have been represented in the task modeling language our code uses, and the behavior reported in terms of what commitments the agents converge to corresponds to what happens in execution of our code, unless otherwise noted.

Let us begin with the simplest problem, from Figure 1. If the agents are initialized with schedules such that A plans to have achieved the enablement, then the agents will converge to agreement over enablement, and achieve their maximum expected reward. This holds true even if B's initial commitment for enablement was null.

If A and B had both begun with null-commitments (because, for example, they did not have any initial schedules from which to extract commitments), then they would stay with these: they are consistent, and neither would benefit from a unilateral change. Similarly, if A had a null-commitment and B had an enables commitment, the consistency procedures would have B revert to a null-commitment, and the agents would again be in agreement. These last examples illustrate how, even for this simple problem, hill-climbing can reach one of two local optima. Finding the global optimum can require a perturbation from the initial commitments to put the agents into a more favorable basin of attraction in the hill-climbing search.

As a second example, consider the variation of the problem in Figure 1 where Method-A2 also enables a method for

C. A quick-and-dirty policy cannot be generated that reaches final states satisfying both commitments, and so the commitments extracted from the policy will revert one of the commitments to a null-commitment (which one is reverted depends on the relative rewards and probabilities associated with the methods). In the subsequent round of distributed hill-climbing, the null-commitment by A (the source) will cause the agent at the target end of the null-commitment to change to a null-commitment, and the three agents will converge to commitments where one of the enablements is committed to and the other is not.

As a final example, Figure 6 shows a problem that is like Figure 1, except between A and B there is a C, creating an enablement chain from A to C to B. As before, each agent also has a local task competing with its task in the chain. Further, let's assume that the expected quality of Method-A1 for A and Method-C1 for C is higher than for Method-A2 and Method-C2, respectively. Finally, assume that there is no initial schedule.

In this case, A and C recognize the first coordination opportunity, and C and B recognize the second. Lacking an initial schedule, the agents initialize their commitments for the coordination opportunities to null-commitments. They exchange these, and find that they are all consistent, and so could represent a valid set of agreements.

However, when the agents formulate their quick-and-dirty local policies and extract commitments, A determines that it actually plans to perform its enablement action. (Note that the null-commitment to the enablement does not bias A away from this action, but rather just does not bias it toward this action. Since Method-A1 is the best choice for purely local reasons, A plans to execute it.) Therefore, when the agents exchange their potentially-revised commitments, A announces that it has indeed changed one of its commitments.

As a result, the agents iterate by making the new commitments consistent. This in turn causes C to change its null-commitment for the enablement opportunity with A into a enablement commitment. The agents then once again formulate policies using quick-and-dirty models, and extract commitments from these. Now, C has changed a commitment, because it has found that it serendipitously is now doing the enablement source for B. The commitments are exchanged, along with the information that some agent has changed a commitment. This ultimately leads the agents to finish the enablement chain: even though the agents began with consistent commitments to not help each other at all, the distributed hill-climbing procedure has reached the optimum set of commitments corresponding to each link in the chain of enablements being done.

Conclusion and Future Directions

In this paper, we have described an approach to coordinating the activities of multiple agents that works in stochastic problem domains by combining high-level agreements over inter-agent commitments with detailed local policy formulation by each of the agents. The result is an approach that gives agents individual flexibility to optimize their local performance, while biasing their local policies to fulfill the commitments to other agents to which they have agreed. This allows the problems of making coordination commitments and of formulating local policies to be largely decoupled, allowing each to be solved much more quickly than solving them together by formulating a full joint policy. A major contribution of this paper is in describing a process for casting commitments into the local MDP models of agents to achieve this kind of bias. While we illustrated this approach in the context of biasing agents to act so as to enable actions for others, we have applied the same principles in this paper to a variety of other kinds of interactions, including commitments *not* to disable the possible actions of others, and syncrhonization. We also require agents to be able to choose commitments without examining in detail every possible combination of policies that they could adopt. As we have shown, agents that agree on commitments tend to do better than agents that disagree, even if the agreed-upon commitments are not the best possible. So, at the core of the agents' coordination protocol is the process of ensuring agreement over consistent sets of commitments. Another contribution is demonstrating how the problem of converging on commitments can be solved by interleaving making tentative agreements with using quick-and-dirty task models to assess feasibility and desirability of the agreements.

The techniques that we have described have all been implemented as part of a larger effort under the DARPA COORDINATORs program. These techniques have been applied to problems ranging up to nearly 100 agents, where each agent can have dozens of methods and methods have uncertainty in durations and outcomes that lead to huge state spaces. Applying MDP techniques to these problems has only been possible due to the techniques that we describe here.

We see a number of directions in which these techniques could be improved. One is to develop more principled techniques for assigning rewards to bias agents. One thread of related research has looked at other ways of biasing the policy formulation process by imposing constraints on the policy itself rather than modifying the states (Witwicki & Durfee 2007). Another direction is to exploit other ideas for choosing initial commitment values. A third direction is improving the heuristics for estimating the global value of particular commitment combinations based on local expected utility. We would also like to consider the dynamics of the commitments themselves, allowing agents to revisit commitment decisions as circumstances unfold.

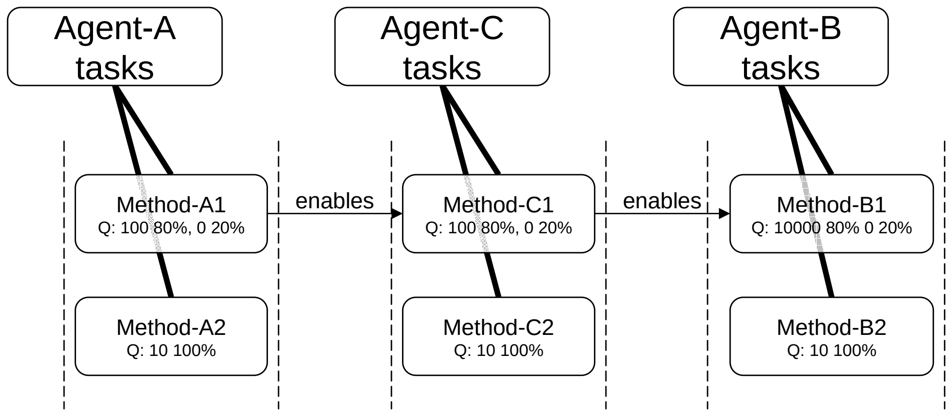

Figure 6: This simple 3-agent additive task model represents a problem where there is a multi-link enablement chain, and where each agent would prefer doing its method in the chain if it knows that that action is enabled.

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.

References

Becker, R.; Zilberstein, S.; Lesser, V.; and Goldman, C. V. 2004. Solving transition-independent decentralized markov decision processes. *JAIR* 22:423–455.

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

Clement, B. J.; Durfee, E. H.; and Barrett, A. C. 2007. Abstract reasoning for planning and coordination. *JAIR* 28:453–515.

Goldman, C. V., and Zilberstein, S. 2004. Decentralized control of cooperative systems: Categorization and complexity analysis. *JAIR* 22:143–174.

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*.

Musliner, D.; Goldman, R. P.; Boddy, M.; Durfee, E.; and Wu, J. 2007. "Unrolling" complex task models into MDPs. In *Proceedings of the AAAI Spring Symposium on Game Theoretic and Decision Theoretic Agents*.

Nair, R.; Pynadath, D.; Yokoo, M.; Tambe, M.; and Marsella, S. 2003. Taming decentralized MDPs: Towards efficient policy computation for multiagent settings. In *Proceedings of IJCAI*.

Nau, D.; Ghallab, M.; and Traverso, P. 2004. *Automated Planning: Theory & Practice*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

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

Witwicki, S., and Durfee, E. H. 2007. Commitment-driven distributed joint policy search. In *Proceedings of AAMAS*, 480–487.

Wu, J., and Durfee, E. H. 2007. Solving large TAEMS problems efficiently by selective exploration and decomposition. In *Proceedings AAMAS*, 291–298.