Decentralized Markov Decision Processes with Event-Driven Interactions

Raphen Becker, Shlomo Zilberstein, Victor Lesser
University of Massachusetts
Department of Computer Science
Amherst, MA 01003
{raphen,shlomo,lesser}@cs.umass.edu

Abstract

Decentralized MDPs provide a powerful formal framework for planning in multi-agent systems, but the complexity of the model limits its usefulness. We study in this paper a class of DEC-MDPs that restricts the interactions between the agents to a structured, event-driven dependency. These dependencies can model locking a shared resource or temporal enabling constraints, both of which arise frequently in practice. The complexity of this class of problems is shown to be no harder than exponential in the number of states and doubly exponential in the number of dependencies. Since the number of dependencies is much smaller than the number of states for many problems, this is significantly better than the doubly exponential (in the state space) complexity of DEC-MDPs. We also demonstrate how an algorithm we previously developed can be used to solve problems in this class both optimally and approximately. Experimental work indicates that this solution technique is significantly faster than a naive policy search approach.

1. Introduction

Markov decision processes (MDPs) provide a foundation for much of the work on stochastic planning. The rapid progress in this area has led researchers to study extensions of the basic model that are suitable for multi-agent systems. Examples of such extensions include the Multi-agent Markov Decision Process (MMDP) proposed by Boutilier [3], the Partially Observable Identical Payoff Stochastic Game (POIPSG) proposed by Peshkin et al. [10], the multi-agent decision process proposed by Xuan and Lesser [15], the Communicative Multi-agent Team Decision Problem (COM-MTDP) proposed by Pynadath and Tambe [11], the Decentralized Markov Decision Process (DEC-POMDP and DEC-MDP) proposed by Bernstein et al. [2], and the DEC-POMDP with Communication (Dec_Pomdp_Com) proposed by Goldman and Zilberstein [5].

Recent studies show that decentralizing knowledge and control among multiple agents has a large impact on the complexity of planning. Specifically, the complexity of both DEC-POMDP and DEC-MDP has been shown to be NEXP-complete, even when only two agents are involved [2]. This is in contrast to the best known bounds for MDPs (P-complete) and POMDPs (PSPACE-complete) [9, 8]. Recent studies of decentralized control problems (with or without communication between the agents) have confirmed that solving even simple problem instances is extremely hard [11, 14]. And while a general dynamic programming algorithm for solving DEC-POMDPs has been recently developed [6], it is unclear whether the algorithm can solve large realistic problems.

The complexity of the general problem of solving DEC-POMDPs has generated a growing interest in special classes that arise in practice and have properties that reduce the overall complexity and facilitate the design of algorithms. One aspect of decentralized control that makes it so difficult is the partial transfer of information between the agents via their observations or explicit communication. When the agents can fully share their observations every step, the problem resembles an MMDP [3] and can be reduced to an MDP, which has polynomial complexity. In previous work [1], we have identified a class of problems in which the agents gained no information about each other while executing a plan. The interaction between the agents occurred through a component of the reward function, which neither agent could observe. We showed that the complexity of that model is exponential.

Interesting practical problems typically allow for some interaction between the agents while executing their plans. This work identifies a model that allows for a structured form of information transfer between the agents. The complexity of this class of problems is doubly exponential in the level of interaction between two agents, and exponential in the non-interacting parts of the decision process. The class of problems we focus on in this paper involve two cooper-

AAMAS'04, July 19-23, 2004, New York, New York, USA.

ating agents, each having its own local set of tasks to perform, with a specific structured interaction between them. This interaction allows the outcome of actions performed by one agent to depend on the completion of certain tasks by the other agent. This form of interaction has been studied extensively within the multi-agent community [7, 12]. Some instances of this type of interaction that have been previously studied are enables/facilitates interrelationships, whereby one agent executing a task enables the other agent to execute another task, or it may increase the likelihood of success. Another example is a non-consumable resource, which one agent can lock and thus prevent the other agent from using.

In addition to the naive approach (complete policy search) to solving a problem like this, we introduce a mapping that allows the optimal policy to be found using our previously developed Coverage Set Algorithm [1]. The key idea is that given a policy for one agent we can construct an MDP to find the corresponding optimal policy for the other agent. This new *augmented* MDP is constructed based on a set of parameters computed from the fixed policy. The Coverage Set Algorithm takes this augmented MDP and finds the set of optimal policies for all possible parameter values. This set includes the optimal joint policy. Our experimental results show that this solution technique performs much better than the naive algorithm.

Section 2 provides a formal description of the problem, illustrated with an example in the TAEMS task description language. In section 3, we identify an upper-bound on the complexity of this class of problems by examining the running time of the complete policy search. Complete policy search is not a good way to solve problems of this type, however, and in section 4 we discuss an alternative algorithm. Section 5 demonstrates the performance of this algorithm on a problem modelled in TAEMS. The contributions of this work and directions for further work are summarized in section 6.

2. Formal Definition of the Model

Intuitively, the special class of problems we focus on involves two agents, each having a "local" decision problem or task that can be modeled as a standard MDP. This local decision problem is fully observable by the agent. The agents interact with each other via a set of structured transition dependencies. That is, some actions taken by one agent may affect the transition function of the other agent.

To illustrate this concretely, we present a problem in TAEMS [4]. TAEMS is a hierarchical task modeling language that has been used successfully in a number of real systems [13]. Figure 1 is an example task structure. In it, the two agents each have one task: T_1 for agent 1 and T_2 for agent 2. Both of those tasks can be decomposed into two

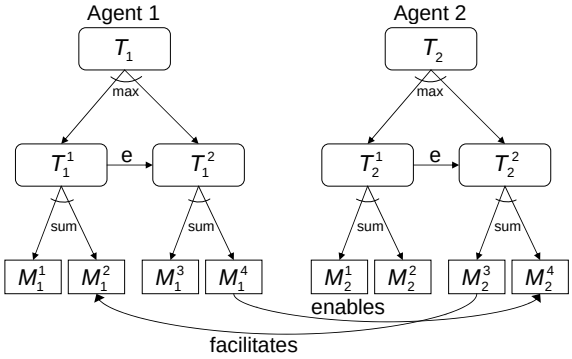

Figure 1. An example TAEMS task structure

subtasks, i.e. $T_1 \rightarrow T_1^1$ and T_1^2. Each of the subtasks is decomposed into two methods, i.e. $T_1^1 \rightarrow M_1^1$ and M_1^2. Methods are the atomic units of a task and the agents can execute them. Executing a method takes *time* and produces *quality*, over some distribution. A quality of 0 represents a method that has not been successfully executed. Tasks accumulate quality from their children in many different ways. Two are shown in the example: sum and max. The quality of T_1^1 is the sum of the qualities of its children, and the quality of T_1 is the max quality of its children. The goal of the system is to maximize the sum of the qualities of the highest level tasks in both agents before the deadline.

The two agents do not operate independently, however. TAEMS has three different types of interrelationships between agents, only one of which is used in this example: facilitates/enables. This type of interrelation is a temporal constraint: M_1^4 must be executed successfully *before* M_2^4 is executed for M_2^4 to produce a nonzero quality. Facilitates is similar though less severe. If M_2^3 is successfully executed before M_1^2, then M_1^2 is *more likely* to produce a higher quality than if it was not facilitated. These are examples of the type of dependencies allowed between agents in our model, though it is not limited to interrelations of this nature.

To define these kind of dependencies more formally, we need to first introduce the general model of decentralized MDPs [2].

Definition 1 *A 2-agent* **DEC-MDP** *is defined by a tuple* $\langle S, A, P, R, \Omega, O \rangle$*, where*

- S *is a finite set of world states, with a distinguished initial state* s^0.
- $A = A_1 \times A_2$ *is a finite set of actions.* A_i *indicates the set of actions taken by agent* i.
- P *is a transition function.* $P(s'|s, a_1, a_2)$ *is the probability of the outcome state* s' *when the action pair* a_1, a_2 *is taken in state* s.

AAMAS'04, July 19-23, 2004, New York, New York, USA.

- *R is a reward function. $R(s, a_1, a_2, s')$ is the reward obtained from taking actions a_1, a_2 in state s and transitioning to state s'.*
- *Ω is the set of all observations for each of the agents.*
- *O is the observation function. $O(s, a_1, a_2, s', o_1, o_2)$ is the probability of agents 1 and 2 seeing observations o_1, o_2 respectively after the sequence s, a_1, a_2, s' occurs.*
- *Joint full observability: the pair of observations made by the agents together fully determine the current state. If $O(s, a_1, a_2, s', o_1, o_2) > 0$ then $P(s'|o_1, o_2) = 1$.*

Definition 2 *A* **factored***, 2-agent DEC-MDP is a DEC-MDP such that the world state can be factored into two components, $S = S_1 \times S_2$.*

Factoring the state space of a DEC-MDP could be done in many ways. The intention of such a factorization is a separation of components of the world state that belong to one agent versus the other. This separation does not have to be strict, meaning that some components of the world state may not belong to either agent and could be included in both. For example, time could be a component of S_1 and S_2.

We refer to s_i, a_i and o_i–the components of the factored DEC-MDP that apply to just one agent–as the *local* states, actions, and observations for agent i. Just as in a DEC-MDP, a local policy (or just policy) for one agent is a mapping from sequences of observations to local actions (we will simplify this later). A **joint policy** is a set of policies, one for each agent.

Our TAEMS example can be represented by a factored DEC-MDP:

- The local state of each of the agents is composed of the current time and the qualities of each of the methods.
- The actions for the agents are to execute one of their methods.
- The transition function is based on the time/quality distribution for the methods the agents choose to execute, taking into account the facilitates/enables interrelationships.
- The reward is only received in a terminal state, and it represents the sum of the qualities of the highest level tasks at that time.
- Each agent fully observes is own local state. In addition, when an agent attempts to execute a constrained method it learns whether the interrelationship was satisfied.

What is interesting about factoring a DEC-MDP is not that the world state can be factored but the properties that a particular factorization have. In this problem we are looking for a factorization that minimizes the interaction between the two agents. The next two definitions introduce some interesting properties that the factored, 2-agent DEC-MDPs we are working with have.

Definition 3 *A factored, 2-agent DEC-MDP is said to be* **locally fully observable** *if $\forall o_i \; \exists s_i \; : P(s_i|o_i) = 1$.*

That is, each agent fully observes its own local state at each step. Note that Ω and O are now redundant in the definition of a factored 2-agent DEC-MDP with local full observability and can be removed.

Definition 4 *A factored, 2-agent DEC-MDP is said to be* **reward independent** *if there exist R_1 and R_2 such that*

$$R((s_1, s_2), (a_1, a_2), (s'_1, s'_2)) = R_1(s_1, a_1, s'_1) + R_2(s_2, a_2, s'_2)$$

That is, the overall reward is composed of the sum of the local reward functions, each of which depends only on the local state and action of one of the agents. In the example, the reward is received only in the terminal states and it is additive between tasks and across agents, therefore the reward function for this factorization is reward independent.

Next we define the interaction between the two agents as event-driven, meaning that an event in one agent influences an event in the other agent.

2.1. Event-Driven Interactions

In this section we will fully define the transition function. The basic idea is that transitions can take two forms. First, many local transitions for one agent are independent of the other agent, which means that they depend only on the local state and action. However, the interrelationships between the agents mean that some transitions depend on the other agent. This interaction is described by a dependency and the change in transitions that result when the dependency is satisfied. We start by defining events, which form the basis of the dependency.

Definition 5 *A* **primitive event***, $e = (s_i, a_i, s'_i)$ is a triplet that includes a state, an action, and an outcome state. An* **event** *$E = \{e_1, e_2, ..., e_h\}$ is a set of primitive events.*

The history for agent i, $\Phi_i = [s_i^0, a_i^0, s_i^1, a_i^1, ...]$ is a valid execution sequence that records all of the local states and actions for one agent, beginning with the local starting state for that agent. A primitive event $e = (s_i, a_i, s'_i)$ occurs in history Φ_i, denoted $\Phi_i \models e$, iff the triplet (s_i, a_i, s'_i) appears as a subsequence of Φ_i. An event $E = \{e_1, e_2, ..., e_h\}$ occurs in history Φ_i, denoted $\Phi_i \models E$ iff $\exists e \in E \; : \; \Phi_i \models e$.

Events are used to capture the fact that an agent did some specific activity like execute a method that enables a remote task. In some cases a single local state may be sufficient, but because of the uncertainty in the domain that method could be executed from many different states (different current times, different current qualities for the other methods)

AAMAS'04, July 19-23, 2004, New York, New York, USA.

we generally need a set of primitive events to capture an activity.

An example of an event would be successfully executing M_1^4 before time 10. It would be composed of primitive events of the form $(time < 10, q_1^1, q_1^2, q_1^3, q_1^4 = 0)$, *execute* M_1^4, $(time < 10, q_1^1, q_1^2, q_1^3, q_1^4 > 0)$, where q_i^k is the current quality of method M_i^k.

Definition 6 *A primitive event is said to be* **proper** *if it can occur at most once in any possible history of a given MDP. That is* $\forall \Phi = \Phi^1 e \Phi^2 \ : \ \neg(\Phi^1 \models e) \wedge \neg(\Phi^2 \models e)$. *An event* $E = \{e_1, e_2, ..., e_h\}$ *is said to be* **proper** *if it consists of mutually exclusive proper primitive events with respect to some given MDP. That is:*

$$\forall \Phi \ \neg \exists i \neq j \ : \ (e_i \in E \wedge e_j \in E \wedge \Phi \models e_i \wedge \Phi \models e_j)$$

We limit the discussion in this paper to proper events because they are sufficient to express the desired behavior and because they simplify discussion. For a discussion on how non-proper events can be made proper see [1].

The event *executing* M_1^4 *before time* 10 is proper because the primitive events that compose the event include the transition from $q_1^4 = 0$ to $q_1^4 > 0$. Since the quality of a task is always nondecreasing, this transition can never occur twice.

The interaction between the agents takes the form of a triggering event in agent i and a set of state-action pairs for agent j that is affected. This interaction is called a dependency.

Definition 7 *A* **dependency** $d_{ij}^k = \langle E_i^k, D_j^k \rangle$, *where* E_i^k *is a proper event defined over primitive events for agent* i, *and* D_j^k *is a set of unique state-action pairs for agent* j. *Unique means* $\neg \exists k, k', s_j, a_j$ *s.t.* $\langle s_j, a_j \rangle \in D_j^k \wedge \langle s_j, a_j \rangle \in D_j^{k'} \wedge k \neq k'$.

Definition 8 *A dependency* d_{ij}^k *is* **satisfied** *when* $\Phi_i \models E_i^k$. *Boolean variable* $b_{s_j a_j}$ *is* true *if the related dependency is satisfied and* false *if it is not satisfied or there is not a related dependency:*

$$b_{s_j a_j} = \begin{cases} \text{true} & \exists k, \ s.t. \ \langle s_j, a_j \rangle \in D_j^k \wedge \Phi_i \models E_i^k \\ \text{false} & otherwise \end{cases}$$

Definition 9 *A* **transition function** *for event driven interactions is divided into two functions,* P_i *and* P_j. *They define the distribution* $P_i(s_i' | s_i, a_i, b_{s_i a_i})$.

When an agent takes an action that could be influenced by a dependency it learns the status of that dependency, whether or not it was satisfied (i.e. whether the task was facilitated). The idea is that an agent knows why things happened after they do. For example, if an agent attempts to execute a task that has not been enabled, it realizes that it does not have the data necessary for the task when it fails. An argument can be made that the agent should be able to check whether it has the available data before it attempts to execute the task. This can be accomplished by a 'free' action that reveals the status of the dependency. Essentially, the dependency modifies the transition for the free action in addition to facilitating the task.

Dependency $d_{1,2}^{10}$ is the dependency that represents method M_2^4 having been enabled when agent 2 attempts to execute it at time 10. The event E_1^{10} is the event described earlier where the enabling method M_1^4 has finished executing successfully before time 10. D_2^{10} contains state-action pairs representing agent 2 attempting to execute M_2^4 at time 10. There is exactly one dependency of this type for each time that agent 2 could attempt to execute method M_2^4. If agent 1 successfully executes M_1^4 at time 6, then all of the dependencies $d_{1,2}^t$ where $t > 6$ will be satisfied (by the same primitive event in agent 1), but each of those satisfied dependencies modify a different set of probabilities in agent 2. Agent 2 can attempt M_2^4 at time 10 and again at time 14, and both times the method will be enabled but through different dependencies ($d_{1,2}^{10}$ and $d_{1,2}^{14}$ respectively).

2.2. Defining the Policy

While we have defined a local state space, action space, transition function and reward function for each agent, this unfortunately does not define a local MDP for each agent. The reason is because the local state is not Markov. When an agent learns the status of a dependency, it changes its belief about the state of the other agent and the impact of future dependencies. This information is contained within the history of an agent, but not in the previously defined local state. Therefore, the local state S_i of agent i must be modified to include the dependency history (i.e. at time t dependency d_{ij}^k was satisfied). This modified local state is S_i'. $\langle S_i', A_i, P_i, R_i \rangle$ does define a local MDP, and the local policy for agent i is $\pi_i : S_i' \rightarrow A_i$.

When one of the TAEMS agents attempts a task with an incoming dependency, its next observation includes whether that dependency was satisfied (i.e. was that task enabled). That information is not stored in the local state of the agent that we defined earlier because it was not necessary in the DEC-MDP (transitions in the DEC-MDP are defined over world states not local states). However, that knowledge changes this agent's belief about the other agent's local state, and that could influence a future expectation of a method being enabled.

In the general case the state will need to include a variable for each dependency, which encodes all of the information gained about that dependency. For specific problems, however, that information is often redundant. For example, in TAEMS the state needs to include the time the outgoing interrelations are enabled, the last time an incoming interrelation was not enabled and the first time it was enabled.

AAMAS'04, July 19-23, 2004, New York, New York, USA.

This information completely encodes everything an agent knows about the dependencies. It is important to notice that in the TAEMS case information is recorded per interrelationship, not per dependency. This is because each interrelationship has multiple dependencies associated with it.

2.3. Evaluating a Joint Policy

The value function that we are trying to maximize is the original value function for the DEC-MDP. However, evaluating a pair of policies is much easier on a new DEC-MDP constructed from the expanded local states and new transition function because the policies are not defined over the same state space as the original DEC-MDP.

- The states $S' = S'_1 \times S'_2$.
- The actions $A = A_1 \times A_2$.
- The transition function $P = P_1 \times P_2$.
- The reward function $R = R_1 + R_2$.
- The observations for each agent are its local component of the state.

The expected reward the system will receive given a pair of policies can be found by running policy evaluation as if this was an MDP because there is a direct mapping from world state to joint action.

3. Problem Complexity

An upper bound on the complexity of the DEC-MDP with event driven interactions can be derived from the complexity of complete policy search.

Theorem 1 *A DEC-MDP with event driven interactions has complexity exponential in $|S|$ and doubly exponential in the number of dependencies.*

Proof In a DEC-MDP with event driven interactions, the number of joint policies is exponential in the number of states because the policy is a mapping from states to actions (unlike the DEC-MDP which is a mapping from sequences of observations to actions). However, the number of states in the new DEC-MDP is exponentially larger in the number of dependencies than the original state space. Therefore, the number of joint policies is exponential in the original state space and doubly exponential in the number of dependencies.

Evaluating a policy can be done with standard policy evaluation for MDPs because in the new DEC-MDP, the joint policy is a direct mapping from world states to joint actions. Policy evaluation for MDPs is polynomial in the number of states, so this does not raise the complexity.

Therefore, the DEC-MDP with event driven interactions has complexity exponential in $|S|$ and doubly exponential in the number of dependencies. ∎

The reason this class of problems is easier to solve than DEC-MDPs is because the size of the policy space has been reduced. It was reduced by separating the part of the history that is necessary to be memorized (interactions between the agents) from the part that is not necessary (local state). Specific problems may have additional structure that further reduces the complexity. For example, in TAEMS the complete interaction history is not necessary to remember so the complexity is doubly exponential in the number of facilitates/enables, not the number of dependencies. If the problem is such that there is an ordering over the interactions and only the most recent interaction must be memorized then the complexity drops to exponential.

4. Solving the Problem

Solving an exponential (or worse) problem using complete policy search is intractable for all problems of moderate or larger size. The reason is that for complete policy search, the worst case complexity is also the best case complexity. That means for every problem every policy must be evaluated, regardless whether it has characteristics that make it a simple problem. Some problems, however, are structured such that many policies are clearly worse than other policies. For example, suppose M_1^4 and M_2^4 have quality outcomes significantly higher than the other methods. A simple analysis of the problem would indicate that nearly any policy that attempts both M_1^4 and M_2^4 has a higher value than any policy that does not. This significantly reduces the number of policies that must be searched.

While the worst case complexity of the Coverage Set Algorithm may be similar to complete policy search, the best case is only polynomial in the state space and exponential in the number of dependencies. The reason the Coverage Set Algorithm has such a variance in complexity is that it is essentially performing a general analysis of the problem. The more obvious a solution, the faster the algorithm runs. However, in the worst case, no useful information is gained through the analysis and it will perform slower than complete policy search due to extra overhead.

4.1. Coverage Set Algorithm (CSA)

The algorithm can be divided into three steps: create the augmented MDP, find the optimal coverage set, search through the set for the optimal joint policy. Each of the three steps is summarized below. For a detailed explanation of the CSA including pseudo-code, see [1].

4.1.1. Augmented MDP The first step is a domain specific step involving translating the problem into a form the CSA works with. This is done by creating an **augmented MDP**. An augmented MDP is a decision problem for agent i that maximizes the global value given a fixed policy for agent j. This may not be the optimal *joint* policy, but it is

AAMAS'04, July 19-23, 2004, New York, New York, USA.

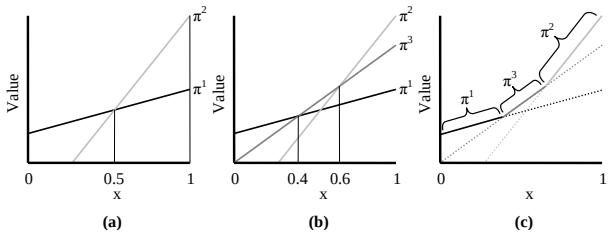

Figure 2. Illustration of the search process in a one dimensional parameter space.

the absolute best policy agent i could choose given the fixed policy for agent j.

An augmented MDP has three properties. First, an augmented MDP is an MDP defined over the states and actions for agent i given a policy π_j for agent j. The transition function and reward function can depend on π_j.

Second, the augmented MDP maximizes the global value of the system for a given policy for agent j. This means that the policy for agent j can be evaluated independently of the policy agent i adopts, and the global value is equal to the independent expected value of agent j's policy plus the expected value of the augmented MDP given both policies: $GV(\pi_i, \pi_j) = V_{\pi_j}(s_j^0) + V_{\pi_i}^{\pi_j}(s_i^0)$.

The third and final property of the augmented MDP is that the value function of the augmented MDP, $V_{\pi_i}^{\pi_j}(s_i^0)$, must be a linear combination of a set of parameters computed from the policy for agent j. Note that any nonlinear function can be made linear by adding additional parameters.

If an augmented MDP can be created for a problem, then the CSA can find the optimal joint policy for that problem.

4.1.2. Optimal Coverage Set The second step of the algorithm corresponds to the analysis of the problem. Each policy for agent j is reducible to a set of parameters with agent i. Each augmented MDP has an optimal policy. That set of optimal policies is the optimal coverage set for agent i. An 'easy' problem is one in which agent i has the same optimal policy for many different agent j policies, and the size of its optimal coverage set is small.

The optimal coverage set can be viewed geometrically as a piecewise-linear and convex surface over the parameters taken from agent j's policies. The pieces of the surface are composed of the value functions of the augmented MDPs with their corresponding optimal policies. They are found through a search process, illustrated in Figure 2.

First, optimal policies at the boundaries of the parameter space are found using dynamic programming on the augmented MDPs instantiated by $x = 0$, (π^1), and $x = 1$, (π^2), Figure 2(a). Next, those lines are intersected and a new optimal policy π^3 is found for $x = 0.5$, Figure 2(b). The intersections between those three lines yield two new points, 0.4 and 0.6. Since no new optimal policies are discovered at those two points, the search is finished and the three optimal policies found form the optimal coverage set.

4.1.3. Optimal Joint Policy The final step in the algorithm is to find an optimal joint policy. One of the policies in the optimal coverage set is part of an optimal joint policy. To find it, the algorithm performs a policy search through the relatively small set. For each policy in the set it finds the corresponding optimal policy for agent j and evaluates the joint policy. The best pair is the optimal joint policy.

4.2. Constructing the Augmented MDP

To show that the CSA can be used for this problem, we must define an augmented MDP.

Let $MDP_i = \langle S_i', A_i, P_i, R_i \rangle$ represent the local MDP for agent i as defined earlier. Let $MDP_i^{\pi_j} = \langle S_i', A_i, P_i', R_i' \rangle$ represent the augmented MDP for a given π_j. The states and actions do not change in the augmented MDP, but the transition function and reward function do. The transition function changes to take into account the incoming dependencies, d_{ji}^k. It is modified by the likelihood that an incoming dependency is satisfied in the other agent and the change in probability that dependency incurs.

Definition 10 *$P_i'(s_i'|s_i, a_i)$ is the transition function for the augmented MDP. For the probabilities not altered by an incoming dependency, $P_i'(s_i'|s_i, a_i) = P_i(s_i'|s_i, a_i, false)$. For the others,*

$$\forall k, \langle s_i, a_i \rangle \in D_i^k, P_i'(s_i'|s_i, a_i) = P_i(s_i'|s_i, a_i, false) +$$
$$P(d_{ji}^k|s_i) \left[P_i(s_i'|s_i, a_i, true) - P_i(s_i'|s_i, a_i, false) \right],$$

where $P(d_{ji}^k|s_i)$ is the probability that dependency d_{ji}^k is satisfied given that agent i is in state s_i.

The reward function is modified to incorporate the changes in expected value the other agent receives when outgoing dependencies are satisfied.

Definition 11 *$R_i'(s_i, a_i, s_i')$ is the reward function for the augmented MDP. For each primitive event $e = \langle s_i, a_i, s_i' \rangle$ that does not satisfy an outgoing dependency, $R_i'(s_i, a_i, s_i') = R_i(s_i, a_i, s_i')$. For the others,*

$$\forall k, \langle s_i, a_i, s_i' \rangle \in E_i^k, R_i'(s_i, a_i, s_i') =$$
$$R_i(s_i, a_i, s_i') + V_{\pi_j}^{s_i'}(s_j^0) - V_{\pi_j}^{s_i}(s_j^0),$$

where $V_{\pi_j}^{s_i}(s_j^0)$ is the expected value of the start state of agent j's local MDP given the policy π_j and the dependency history contained in s_i.

The parameters from agent j's policy and MDP take two forms, $P(d_{ji}^k|s_i)$ and $V_{\pi_j}^{s_i}(s_j^0)$. Neither of these is difficult

AAMAS'04, July 19-23, 2004, New York, New York, USA.

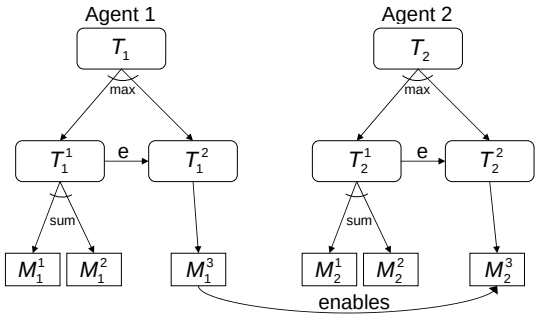

Figure 3. The TAEMS problem structure for the experiments.

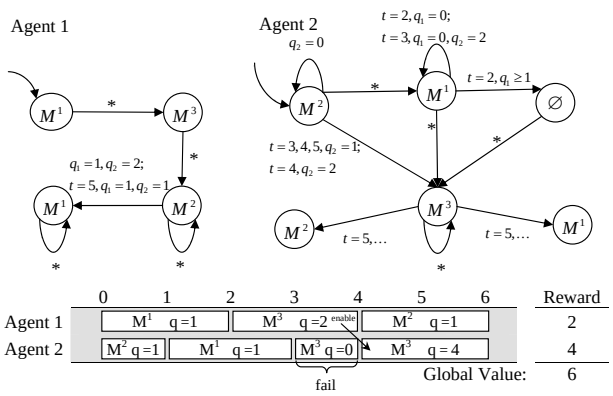

Figure 4. The optimal joint policy and an example execution.

to compute, and they can be derived at the same time. The value V can be obtained by running policy evaluation on the MDP obtained by applying the dependencies satisfied in s_i. At the same time, the probability of reaching each state can be computed. $P(d^k_{ji}|s_i)$ is the sum of the probabilities of all primitive events in E^k_j that have a consistent dependency history with s_i.

A value function for an MDP can be represented as the sum over all primitive events in the MDP, the probability of that primitive event occurring times the reward received at that primitive event. This function can be converted to the form $V = c_0 + c_1x_1 + c_2x_2 + ...$ where x_n is the product of one or more parameters (i.e. $P(d^3_{ji}|s^6)P(d^5_{ji}|s^{13})P(d^5_{ji}|s^{16})$). Partially symbolic policy evaluation would generate a value function with this form. While this function is not linear in the parameters, it is linear in these combinations of parameters. Having a linear value function allows the use of the Coverage Set Algorithm.

5. Experimental Results

To test the algorithm on this class of problems, we implemented the example problem shown in Figure 3, which is a simple variant of Figure 1. This section will examine a typical instance of this problem in detail. The agents had 6 time steps to execute their methods. Each method took between 1 and 3 time steps to complete. Methods M^1 and M^2 produced an integer quality between 0 and 2, while M^3 produced a quality of 0, 2 or 4. The method M^3_2 took 1 time step and produced quality 0 if not enabled. After executing M^3_2 agent 2 knows whether it was enabled or not. The agents received a reward after the time ran out equal to the final quality of their task. The global reward being maximized is the sum of the local rewards: $Max(q^1_1 + q^2_1, q^3_1) + Max(q^1_2 + q^2_2, q^3_2)$.

There are four dependencies that represent M^3_2 being enabled at times 2 through 5 (2 is the first time it could be enabled and 6 is the deadline). The state of each agent also includes information about these dependencies. Agent 1 keeps track of the last dependency that was satisfied (for a total of 945 states). Agent 2 keeps track of the last dependency that was not satisfied and the first dependency that was (for a total of 4725 states).

The dimensionality of the search was also different for the two agents because they are on different sides of the dependencies. Agent 1, being the enabler, had parameters that represented the expected value for agent 2 given that agent 1 enabled at time t. There were four different times that agent 1 could enable agent 2 and another parameter for when agent 2 was never enabled leading to a parameter space of size five. Agent 2, being on the receiving end, depends on the probability that it was enabled given the current time and the last time it learned it was not enabled. There were ten probabilities, but the parameter space depended on combinations of those probabilities and was much higher. Since agent 1 had a much lower dimensionality, we chose to find its optimal coverage set.

Figure 4 shows an FSM representation of the optimal joint policy for one instance of the problem. The vertices are the actions taken and the edges are transitions based on the current state. The * matches any state not matched by a more specific label. The optimal coverage set for agent 1 contained 141 policies and took less than an hour to find on a modern desktop computer.

While the complexity of these decentralized MDPs with event-driven interactions is significantly easier than models like the DEC-MDP, it is still intractable for large problems and certain hard small problems. Fortunately, it turns out that the coverage set algorithm is naturally an anytime algorithm with some very nice characteristics. Finding an optimal or near optimal solution usually takes very little time. Proving that the solution is optimal takes the majority of the

AAMAS'04, July 19-23, 2004, New York, New York, USA.

computation. For example, the expected value of the optimal joint policy in Figure 4 is 5.8289. The first joint policy found had a value of 5.6062, or 96.2% of optimal. The optimal joint policy was discovered after only 0.004% of the total computation. The result is a good anytime solution for problems too large to solve optimally.

6. Conclusion

The DEC-MDP framework has been proposed to model cooperative multi-agent systems in which agents receive only partial information about the world. Computing the optimal solution to the general class is NEXP-complete, and with the exception of [6] the only known algorithm is brute force policy search. We have identified an interesting subset of problems that allows for interaction between the two agents in a fixed, structured way. For this class of problems we have identified that the complexity reduces from being doubly exponential in the state space to doubly exponential in the level of interaction between the agents and only exponential in the state space. Since many problems have a level of interaction significantly lower than the number of states, the savings can be quite substantial.

We also provided a mapping to an algorithm that runs much faster than complete policy search. While the high complexity still makes it intractable for large problems, this work does facilitate finding optimal solutions for larger problems that have only a limited amount of interaction between the agents. That can be useful in establishing a baseline for evaluation of approximation algorithms.

The augmented MDP enables a simple yet powerful hill-climbing algorithm that converges on a local optimum. In addition, the coverage set algorithm is also naturally an anytime algorithm with some promising characteristics. Using these two approximations should facilitate finding good solutions to much larger problems. This remains the subject of future work.

7. Acknowledgments

This work was supported in part by NASA under grant NCC 2-1311 and by NSF under grant IIS-0219606. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of NASA or NSF.

References

[1] Raphen Becker, Shlomo Zilberstein, Victor Lesser, and Claudia V. Goldman. Transition-independent decentralized Markov decision processes. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi Agent Systems*, pages 41–48, Melbourne, Australia, July 2003. ACM Press.

[2] Daniel S. Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27(4):819–840, November 2002.

[3] Craig Boutilier. Sequential optimality and coordination in multiagent systems. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 478–485, 1999.

[4] Keith Decker and Victor Lesser. Quantitative modeling of complex environments. *International Journal of Intelligent Systems in Accounting, Finance and Management. Special Issue on Mathematical and Computational Models and Characteristics of Agent Behaviour*, 2:215–234, January 1993.

[5] Claudia V. Goldman and Shlomo Zilberstein. Optimizing information exchange in cooperative multi-agent systems. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi Agent Systems*, pages 137–144, Melbourne, Australia, July 2003. ACM Press.

[6] Eric Hansen, Daniel S. Bernstein, and Shlomo Zilberstein. Dynamic programming for partially observable stochastic games. To appear in *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, 2004.

[7] Nicholas R. Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, 75(2):195–240, 1995.

[8] Martin Mundhenk, Judy Goldsmith, Christopher Lusena, and Eric Allender. Complexity of finite-horizon Markov decision process problems. *Journal of the ACM*, 47(4):681–720, 2000.

[9] Christos H. Papadimitriou and John Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[10] Leonid Peshkin, Kee-Eung Kim, Nicolas Meuleau, and Leslie P. Kaelbling. Learning to cooperate via policy search. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 489–496, San Francisco, CA, 2000. Morgan Kaufmann.

[11] David V. Pynadath and Milind Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research*, 16:389–423, 2002.

[12] Milind Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

[13] Thomas Wagner, Valerie Guralnik, and John Phelps. TAEMS agents: Enabling dynamic distributed supply chain management. *Journal of Electronic Commerce Research and Applications*, 2003.

[14] Ping Xuan and Victor Lesser. Multi-agent policies: From centralized ones to decentralized ones. *Proceedings of the First International Joint Conference on Autonomous Agents and Multi Agent Systems*, pages 1098–1105, 2002.

[15] Ping Xuan, Victor Lesser, and Shlomo Zilberstein. Communication decisions in multi-agent cooperation: Model and experiments. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 616–623, Montreal, Canada, January 2001. ACM Press.