What's Worth Memorizing: Attribute-based Planning for DEC-POMDPs

Christopher Amato and Shlomo Zilberstein
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
{camato,shlomo}@cs.umass.edu

Abstract

Current algorithms for decentralized partially observable Markov decision processes (DEC-POMDPs) require a large amount of memory to produce high quality plans. To combat this, existing methods optimize a set of finite-state controllers with an arbitrary amount of fixed memory. While this works well for some problems, in general, scalability and solution quality remain limited. As an alternative, we propose remembering some attributes that summarize key aspects of an agent's action and observation history. These attributes are often simple to determine, provide a well-motivated choice of controller size and focus the solution search on important components of agent histories. We show that for a range of DEC-POMDPs such attribute-based representation improves plan quality and scalability.

Introduction

Decentralized partially observable Markov decision processes (DEC-POMDPs) provide a powerful and attractive way to model multiagent planning problems under uncertainty which are both sequential and cooperative. As an extension of partially observable Markov decision processes (POMDPs), DEC-POMDPs allow a set of agents to affect a shared objective function and global state while allowing each agent to determine its action using solely local information. In addition to uncertainty about the global state of the problem, each agent must cope with imperfect information about the knowledge and actions of the other agents. These general assumptions allow DEC-POMDPs to model a wide range of problems in cooperative multiagent systems.

Several exact and approximate methods have been developed for solving DEC-POMDPs (Amato *et al.* 2007b; Bernstein *et al.* 2005; Carlin and Zilberstein 2008; Emery-Montemerlo *et al.* 2004; Hansen *et al.* 2004; Nair *et al.* 2003; Petrik and Zilberstein 2007; Oliehoek *et al.* 2008; Seuken and Zilberstein 2007; Szer *et al.* 2005; Szer and Charpillet 2005). Of these, only three algorithms address the infinite-horizon problem (Amato *et al.* 2007b; Bernstein *et al.* 2005; Szer and Charpillet 2005), all of which are approximate. Additionally, an algorithm that can solve a subclass of infinite-horizon DEC-POMDPs has been developed using an average reward formulation (Petrik and Zilberstein 2007). Unfortunately, this approach is restricted to *transition independent* problems with *full local observability*.

Given the high complexity results of DEC-POMDPs (Bernstein *et al.* 2000), one of the main barriers for algorithms is the large amount of memory often required to provide high quality plans. For instance, the optimal algorithm for solving the infinite-horizon problem (Bernstein 2005) evaluates increasingly longer histories of actions and observations until the policy's value no longer changes. This near exhaustive search is intractable for all but the smallest problems. To combat this large memory usage, the approximate methods optimize small, fixed-size finite-state controllers. While this increases scalability in some problems, difficulties remain. For instance, choosing the best controller size can be difficult, partly because controller nodes do not have much meaning. Furthermore, existing methods still require a large amount of time and memory for larger problems and controller sizes.

In order to make better use of limited memory, we propose identifying key attributes of an agent's action and observation history and use them to model potential solutions. These attributes may consist of such information as whether another agent has been seen or an estimate of the system state. An agent can then remember these attributes rather than the observation histories themselves. This approach can improve scalability by drastically reducing the memory requirements. To achieve this goal, we propose using both automatic and user defined methods. Our automatic methods use state estimates as attributes, thus permitting an agent to remember its own location or some estimate. When an estimate cannot be calculated or if too many such estimates exist, the attributes can be defined or adjusted by the user. In these cases we can take advantage of human insight and domain knowledge to identify a compact set of attributes. The solution of a DEC-POMDP can then be solved in two steps, determining the attribute-based model and then solving for a policy based on the given representation. In many problems, this approach is likely to improve both scalability and solution quality.

The rest of the paper is organized as follows. We first describe the DEC-POMDP model, the representation of its solution as a finite-state controller and the relevant previous work. We then discuss our attribute-based representation along with how the attributes are used and chosen. We also present a branch and bound algorithm for determining the optimal action choice given an agent's attribute

values. Lastly, experimental results are provided comparing our attribute-based representations to the current state-of-the-art DEC-POMDP approximation algorithms. These results show for a range of problems that our approach consistently produces plans that are higher valued. This suggests that in many DEC-POMDP domains, using attributes can provide a good way to limit memory and increase plan quality.

Background

We begin by reviewing the DEC-POMDP framework as well as how to represent an infinite-horizon plan as a finite-state controller. We also review current algorithms for solving infinite-horizon DEC-POMDPs.

The DEC-POMDP model

A DEC-POMDP can be defined with the tuple: $\langle I, S, \{A_i\}, P, R, \{\Omega_i\}, O\rangle$

- I, a finite set of agents
- S, a finite set of states
- A_i, a finite set of actions for each agent, i
- P, a set of state transition probabilities: $P(s'|s, \vec{a})$, the probability of transitioning from state s to s' when the set of actions $\vec{a}$ are taken by the agents
- R, a reward function: $R(s, \vec{a})$, the immediate reward for being in state s and agents taking the set of actions $\vec{a}$
- Ω_i, a finite set of observations for each agent, i
- O, a set of observation probabilities: $O(\vec{o}|s', \vec{a})$, the probability of agents seeing the set of observations $\vec{o}$ given the set of actions $\vec{a}$ has been taken which results in state s'

We consider infinite-horizon problems, which involve an unbounded number of steps. At each step, every agent chooses an action based on its local observation history, resulting in an immediate reward and an observation for each agent. Note that because the system state is not directly observed, it may be beneficial for the agent to remember the observation history. A *local policy* for an agent is a mapping from local observation histories to actions while a *joint policy* is a set of policies, one for each agent in the problem. The goal is to maximize the infinite-horizon total cumulative reward, beginning at some initial distribution over states called a *belief state*. Uncertainty about the behavior of other agents can be captured from an agent's perspective using a *generalized belief state*. This is a distribution over not only the states of the problem, but also the current set of policies for the other agents. An important subclass of DEC-POMDPs in which the system state is uniquely defined by the observations of all agents is the DEC-MDP. In a DEC-MDP, if $O(\vec{o}|s, \vec{a}) > 0$ then $P(s'|\vec{o}) = 1$, deterministically identifying the given state. In order to maintain a finite sum over the infinite-horizon, we employ a discount factor, $0 \leq \gamma < 1$.

As a way to model DEC-POMDP policies with finite memory, finite-state controllers provide an appealing option. Each agent's policy can be represented as a local controller and the resulting set of controllers supply the joint policy. A deterministic finite-state controller can formally be defined by the tuple $\langle Q, \psi, \eta\rangle$, where Q is the finite set of controller nodes, $\psi : Q \rightarrow A$ is the action selection model for each node, and $\eta : Q \times A \times O \rightarrow Q$ represents the node transition model for each node given an action was taken and an observation seen. The value for starting in nodes $\vec{q}$ and state s is given by: $V(\vec{q}, s) =$

$$R(s, \vec{a}_{\vec{q}}) + \gamma \sum_{s'} P(s'|\vec{a}, s) \sum_{\vec{o}} O(\vec{o}|s', \vec{a}) V(\vec{q'}_{\vec{q},\vec{a},\vec{o}}, s')$$

This is also referred to as the Bellman equation. The notation $\vec{a}_{\vec{q}}$ represents the set of actions defined by the nodes $\vec{q}$ and $\vec{q'}_{\vec{q},\vec{a},\vec{o}}$ represents the resulting nodes given the previous nodes $\vec{q}$, actions $\vec{a}$ and observations $\vec{o}$. Note that the values can be calculated offline in order to determine controllers for each agent that can then be executed online for distributed control.

Previous work

Approximate algorithms for generating plans for infinite-horizon DEC-POMDPs have been developed by Bernstein *et al.* (2005), Amato *et al.* (2007b) and Szer and Charpillet (2005). The first two methods use linear programming and nonlinear programming (NLP) respectively to determine parameters for stochastic finite-state controllers. As an alternative, Szer and Charpillet use best-first search to construct deterministic controllers. An optimal algorithm has also been developed (Bernstein 2005), but does not perform well in practice due to very large time and memory requirements. As a consequence, we will only discuss the more practical fixed-memory methods.

Bernstein *et al.*'s approach, called bounded policy iteration for decentralized POMDPs (DEC-BPI), improves a set of fixed-size controllers. This is done by iterating through the nodes of each agent's controller and attempting to find an improvement. A linear program searches for a probability distribution over actions and transitions into the agent's current controller that increases the value of the controller for any initial state and any initial node of the other agents' controllers (the generalized belief space). If an improvement is discovered, the node is updated based on the probability distributions found. Each node for each agent is examined in turn and the algorithm terminates when no controller can be improved further. This allows memory to remain fixed, but provides only a locally optimal solution. This is due to the linear program considering the old controller values from the second step on and the fact that improvement must be over all possible states and initial nodes for the controllers of the other agents. As the number of agents or size of controllers grows, this later drawback is likely to severely hinder improvement.

To address these concerns, Amato *et al.* define a set of optimal controllers given a fixed size with a nonlinear program. While it is often impossible to solve general NLPs optimally, many locally-optimal solvers exist. Unlike DEC-BPI, this approach allows start state information to be used so smaller controllers may be generated and all improvement takes place in one step. While concise controllers with high value can be produced with this approach, large controllers result in large NLPs that cannot be solved efficiently with current tools. Because larger controllers may be required for some problems, plan quality suffers in those cases.

Szer and Charpillet have developed a best-first search algorithm that can generate optimal deterministic finite-state controllers of a fixed size. This is done by calculating a heuristic for the controller given the known deterministic parameters and filling in the remaining parameters one at a time in a best-first fashion. This technique generates the optimal deterministic finite-state controller of a given size. But due to poor scalability, it can only produce small controllers, which limits the quality of the resulting plans.

What's worth memorizing?

Because requiring a large amount of memory is often the main bottleneck for DEC-POMDP algorithms, approximate algorithms have focused on fixed-memory solutions. While these algorithms are more scalable than the optimal approach, more must be done in order to improve solution quality. Instead of arbitrarily setting controller size and searching in the space of all possible controllers, we propose using problem dynamics and human knowledge to create a reduced complexity model which can be solved more easily than previous approaches. The model makes use of information about the problem to more intelligently generate a solution with much less memory and time than previous methods.

Our method utilizes a set of attributes rather than entire histories or arbitrary controller nodes in order to generate a plan. That is, because it is both unnecessary and impractical to remember all histories in order to produce a policy, our method seeks to identify a small number of attributes which will help produce a high quality plan. Examples of an attribute may be whether a robot saw a wall after its last action or whether another agent has been observed in the last k steps. These attributes have discrete values associated with them which in this case could be *wall-front*, *wall-left*, *wall-right* and *wall-behind* for the first attribute, and any value from 0 to k indicating the number of steps since the other agent has been seen for the second attribute.

These attributes are essentially landmarks or important pieces of an agent's history. They permit context sensitive policies with attributes as intuitive encapsulations of the observation history. In cases such as when a local or global state can be estimated by an agent, these estimations can be used as attributes. Otherwise, it is often easy for a researcher to identify a set of landmarks based on what she believes will be important for an agent to remember. When generating these attributes, they must be based on the only source of information for each agent: the observations and problem dynamics. Thus, attributes such as the locations of the other agents or the underlying state of the system are not of much use unless the agent possesses some observation that determines these properties. So, if an agent can see another agent when it is close enough or observe some other indicator of the system state (such as the presence or absence of an object), then these types of attributes can be used.

We also require that attribute values are determined by the current attribute, the action taken and the observation seen. An example of this, taken from the two agent tiger problem discussed later, can be seen in Figure 1. In this problem, the agent has three action choices, two possible observations

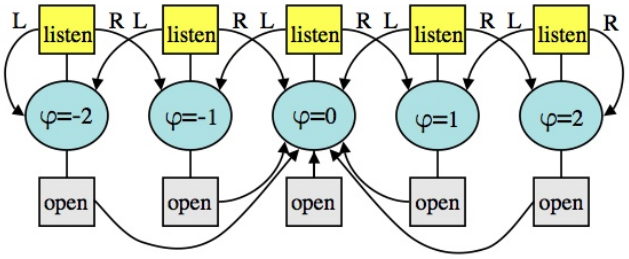

Figure 1: Attribute transitions for the two agent tiger problem. The attribute φ has values between -2 and 2. Actions are shown in squares and transitions are given by the arrows labelled with the relevant observation. If the agent hears the R or L observation after listening, the attribute value moves respectively to the right or left. If the agent chooses an open action, the resulting attribute value is always 0.

and there are two doors each representing a system state. Behind one door is a tiger and behind the other is a large reward. The agents can listen and receive a noisy observation of which door the tiger is lurking behind. Once a door is opened, the tiger is randomly placed behind one of the doors and problem begins again. Using our attribute-based representation, we have one attribute, φ, which begins with value 0, the agent then chooses an action and hears an observation. If a door is opened, the resulting attribute value is always 0 and if the agent listens, the observation heard causes the attribute value to move to the left or right. Once the attribute value is at right or left end, the right and left observations respectively do not change the attribute value any further. This allows the agents to remember how many more times one observation has been heard than the other, up to a constant amount of two in this case.

In this example, the attributes are used by agents with partial observability of the system state and no sharing of information. Because no information is shared, an agent knows when it opens a door, but not when the others do so. This allows each agent to possess some estimate of the system state, but these estimates may be different for each agent. Because there is no common estimate, the solution remains distributed with each agent making an action choice based on local information. Using attributes is an efficient way to remember important parts of this local information and make action choices accordingly. Thus, we can model a class of solutions for a given problem as a finite-state controller and solve for the optimal action choices in many cases. We discuss the details more formally below.

Attribute-based representation

The set of history attributes for an agent is defined as $\Phi = \langle \Phi^1, \ldots, \Phi^n \rangle$, the set of discrete attribute values for each attribute Φ^i. Also, we define a mapping from an agent's history, h, to a vector of attribute values as $\Phi(h) = \langle \varphi^1, \ldots, \varphi^n \rangle$ where each φ^i represents the discrete value of attribute i. We will denote this vector for a given agent i by using $\vec{\varphi}_i$ and a set of such vectors for all agents as $\vec{\varphi}$.

The transition function can be extended to a mapping η^* from a history to an attribute vector for agent i. The mapping is defined inductively as follows. Initially, with no history, the initial attribute vector is used: $\eta^*(\emptyset) = \vec{\varphi}_i^0$. If history h_i is seen followed by the agent taking action a and seeing observation o, $\eta^*([h_i, a, o]) = \eta(\eta^*(h_i), a, o)$. Hence, the transition function η is applied to attribute vector $\eta^*(h_i)$, action a and observation o and a new attribute vector results.

We then say the attribute transitions are *one step consistent* if $\Phi([h_i, a, o]) = \eta(\Phi(h_i), a, o)$. That is, the attribute vector values that result from seeing the sequence of actions and observations in history h up to step i directly followed by action a and observation o on step $i+1$ are equal to the attribute vector values that result from starting with the values mapped to by $\Phi(h_i)$ and transitioning based on taking action a and seeing observation o on the next step. These transitions can be be seen in Figure 1.

Therefore, we consider attribute vector transitions in which value vectors are deterministically updated after each step. While other transitions are possible, this allows a wide range of attributes to be used while permitting them to be represented by a finite-state controller with simple transitions. Also, it is possible to define a stochastic transition function which maps to distributions over nodes, but in this paper we focus on deterministic controllers.

Using an attribute-based representation, a policy for each agent is a mapping from attribute vectors to actions, $\pi_A : \Phi \rightarrow A$. Once such a policy has been defined, we can evaluate a joint policy at state s with current attribute vectors for all agents $\vec{\varphi}$ by:

$$V(s, \vec{\varphi}) = R(s, \vec{a}_{\vec{\varphi}}) + \gamma \sum_{s'} P(s'|s, \vec{a}_{\vec{\varphi}}) \sum_{\vec{o}} P(\vec{o}|s', \vec{a}_{\vec{\varphi}}) V\left(s', \vec{\eta}(\vec{\varphi}, \vec{a}_{\vec{\varphi}}, \vec{o})\right)$$

where $\vec{a}_{\vec{\varphi}}$ defines the actions assigned to the attribute vectors $\vec{\varphi}$ and $\vec{\eta}(\vec{\varphi}, \vec{a}_{\vec{\varphi}}, \vec{o})$ represents the resulting attribute vectors for all agents given by our transition function when the previous set of vectors were $\vec{\varphi}$, actions $\vec{a}_{\vec{\varphi}}$ were taken and observations $\vec{o}$ were seen.

An action mapping π_A is considered optimal in this context if there is no other deterministic assignment of actions to the attribute vectors that produces a higher value given our attribute transition function and the initial state of the problem. This representation is simple, compact and provides a well defined policy for any DEC-POMDP. The resulting policy has the added benefit of being easily readable as actions are chosen based on machine or user defined attribute values.

Producing an optimal mapping

Constructing an optimal action mapping requires searching through the space of action mappings for each agent. If there are $|\Phi|$ different attribute vectors (and thus nodes in each agent's controller), the number of possible action mappings for each agent is $|A|^{|\Phi|}$. Thus, the number of possible action mappings for n agents is $|A|^{n|\Phi|}$. While variations of the algorithms described previously could be used to find action mappings with transitions defined by our attribute-based representation, each approach has significant drawbacks. For instance, DEC-BPI and the NLP approach provide stochastic solutions that are only locally optimal while BFS has limited scalability due to a memory intensive heuristic function.

Algorithm 1: Centralized policy iteration with full observability for a set of agents

input : An assignment of actions to attribute vectors for each agent
output: An upper bound on the value for each state and attribute for each agent

```
begin
    π_0(φ⃗, s) ← curActMap(φ⃗)
    polChange ← true
    t ← 0
    while polchange do
        V_t(φ⃗, s) ← evaluate(π_t)
        t ← t + 1
        polChange ← false
        foreach φ⃗, s do
            π_t(φ⃗, s) ← maxOneStepPol
            if π_t(φ⃗, s) ≠ π_{t−1}(φ⃗, s) then
                polChange ← true
    return V_t
end
```

As an alternative, we present a branch and bound algorithm that, given our representation, can efficiently provide an optimal action assignment. To accomplish this, we order the nodes of the controller and perform a depth first search while evaluating partial action assignments in the controller. That is, we assign an action to the node associated with the current search depth and calculate upper and lower bounds on its value based on assignments to the previous nodes. If the upper bound value of a node is lower than the lower bound of the parent node in our search, we can prune that branch. Thus tight bounds for action assignments can drastically reduce the number of possible action assignments that need to be considered.

We propose a lower bound of random action assignments for unassigned nodes and an upper bound using a centralized policy iteration algorithm. Both of these methods fix the actions of assigned nodes, thus when a controller is fully defined, the lower and upper bounds will both be equal to the actual value of the controller. To find the upper bound, we perform MDP-style policy iteration with a fixed policy for assigned nodes and a centralized policy for unassigned nodes. That is, when the action is not assigned for a given node, we assume the best action is chosen for each agent given the state and the policies of the other agents are known. This approach is shown in Algorithm 1. The policy is initialized with the current action assignment, the resulting controller is evaluated and then the policy is improved by changing the actions for unassigned nodes in order to increase the value of the controller. If no such improvement can be made,

the policy has converged and the value is returned as an upper bound. The improved policy for a set of attribute vectors $\vec{\varphi}$ and a system state s is given by

$$maxOneStepPol = \operatorname*{argmax}_{\vec{a}} \Big[R(s,\vec{a}) + \gamma \sum_{s'} P(s'|s,\vec{a}) \sum_{\vec{o}} P(\vec{o}|s',\vec{a}) V\big(s', \vec{\eta}(\vec{\varphi},\vec{a},\vec{o})\big) \Big]$$

when all attribute vectors $\vec{\varphi}$ do not have assigned actions. If any attribute vector does have an assigned action, the value is only maximized for the other agents. If no attribute vectors have assigned actions, no maximization is performed. This algorithm is an upper bound because value is maximized with the assumption that the state and the values of all agents attributes are known. This also allows different actions to be chosen for each of these attribute values of the agents and states of the system, which relaxes the assumptions necessary for decentralized execution.

This branch and bound approach is an anytime algorithm in that if execution is halted before the optimal assignment is found, the best known mapping at that point is returned. Also, if an optimal action mapping is not an optimal policy, it can be used as an initial policy in other DEC-POMDP algorithms possibly increasing the speed with which an optimal solution is found or otherwise improving the value further by removing the constraints imposed by using attribute values. Additionally, it is worth noting that if a stochastic solution is desired, a modified version of the nonlinear programming method (Amato *et al.* 2007b) can be used. This would allow for stochastic action mappings and may require fewer attribute values, but does not guarantee an optimal mapping is found.

Choosing attributes

In order to encapsulate the relevant information in a problem while retaining the ability to solve for an optimal action mapping, the attributes for our representation must be chosen carefully. If we define attribute values as all possible agent histories, we could represent an optimal policy for a DEC-POMDP, but this may require an infinite number of attribute vectors. Instead, we would like to use a finite number of attribute vectors (and thus have a finite-state controller) to produce a high quality plan. To achieve this goal, we will focus on automated methods based on local or global state estimates and user defined attributes based on domain knowledge.

We will first discuss the generation and use of state estimates. For instance, if the actions and observations of an agent provide enough information, a state estimate can be calculated at each step of the problem. One example is when the observations and system dynamics supply sufficient information about the locations and choices of the other agents, the global state can be estimated. Also, problems in which actions and observations provide information about the agent's own location, but give no information about the other agents allow a local state estimate to be calculated. This structure is seen in DEC-MDPs with independent observations and transitions discussed in (Becker *et al.* 2004).

While remembering the state or estimates of the state limits the amount of memory required to solve a problem, a direct mapping to actions may not be sufficient to represent an optimal policy. Instead, a non-stationary mapping may be necessary due to uncertainty concerning the other agents. This likely depends on how much knowledge each agent has about the others considering all agents know the dynamics of the domain and the problem is solved offline for online execution. Hence agents may know enough about the other agents to produce a more sophisticated policy than that provided by a stationary mapping.

When too many state estimates exist or they cannot be calculated, human knowledge can be utilized. This could be accomplished by deciding when to stop including more state estimates as attributes or otherwise choosing key aspects of the observation history. As pointed out earlier, it is often easy for a researcher to perform such a task. Landmarks such as crossroads or other milestones for an agent can be readily identified in many problems with only cursory knowledge of a domain. Thus, human knowledge can aid in focusing the search to a manageable level while retaining most if not all of the relevant information needed for an agent.

In order to represent states and state estimates in our framework, we can define an attribute with values which correspond to the state seen or estimated for a given agent and these are updated based on the dynamics of the domain. If a set of local states can be defined for an agent, such as the agent's own position, the state estimate can be updated in the same way it is calculated for POMDPs. The probability an agent transitions to state s' given action a was taken and observation o was seen is given by

$$P(s'|a,o) = \frac{\sum_s P(s'|s,a) P(o|s',a) b(s)}{\sum_{s,s'} P(s'|s,a) P(o|s',a) b(s)}$$

Where $b(s)$ is the probability the agent's state was s in the previous step and the other probabilities represent the likelihood of transitioning and observing for an agent with independent transitions and observations.

This process can also be used to generate global state estimates. If an agent's actions and observations provide sufficient information about the state of the system and actions of the other agents, a shared global estimate can be established and the process becomes a POMDP that can be solved centrally. Less information may also allow a global state estimate to be calculated, but sometimes we must settle for an approximation of the global state that is not shared by the set of agents.

The example of the two agent tiger problem discussed earlier is one such case. In this problem each agent can estimate the global state by using local observations. Because the state does not change until an agent opens a door, an observation provides noisy information about the system state. Thus, an agent can compile a finite number of global state estimates by remembering the difference in the number of times an observation is heard up to some value k. Once the agent opens the door, the estimating begins again. While an agent does not know when the other agent opens a door (instead it receives either observation with equal likelihood),

these state estimations provide a useful approximation of the location of the tiger at each step.

Thus, choosing attributes for our representation is an important first step to finding a high quality policy for DEC-POMDPs. The methods based on state estimation can be helpful as even an approximation of the local or global state can provide enough information to select a high quality action. Similarly, many problems have aspects of the agent's histories that a researcher can easily label as useful or useless for an agent to remember. Both approaches can be utilized to represent histories in an intuitive and concise way. In many problems, this representation can then permit an optimal action mapping which will in turn provide a high quality plan.

Experimental results

For this paper, we compared the performance of our attribute-based method against the three state-of-the-art algorithms for infinite-horizon DEC-POMDPs. For each problem, we evaluated how the optimal action mapping given our attribute-based representation performs when compared with the plans generated by the other algorithms. It is worth noting that results from BFS represent optimal deterministic controllers for a given size and the DEC-BPI and NLP results illustrate locally optimal stochastic controllers. The results show the benefit or choosing the given attributes-based representation as well as the fact that an optimal action mapping can be found for these representations in a timely manner.

We solved our representation using the branch and bound method, reporting resulting value, the number of attributes (size of the controller) and running time. Our algorithm was initialized by choosing the best of 10 deterministic controllers with parameters chosen from a uniform distribution. The algorithm was then run until the optimal mapping was found. In each problem, we used a combination of automated and human adjusted attributes, demonstrating that good attributes can be chosen with only small amount of human effort.

For the NLP method, BFS and DEC-BPI results for previously studied domains were taken from (Amato *et al.* 2007a) and we also tested these algorithms on the new domains. There algorithms were run until memory was exhausted or time expired (4 hours). We then report the highest valued results for each method. BFS and our attribute mapping approach were run on a 1.86GHz Intel Core 2 Duo with 3 GB RAM, while DEC-BPI was run on a 3.4GHz Intel Pentium 4 with 2 GB RAM and the NLP method was run on the NEOS server (http://www-neos.mcs.anl.gov/) on unknown systems. Because different machines were required for the algorithms, computation times are not directly comparable. Nevertheless, we expect that they would only vary by a small constant. The results for three test domains (with a discount factor of 0.9) are shown in Table 1.

Two agent tiger problem

The first test domain with 2 states, 3 actions and 2 observations is the two agent tiger problem (Nair *et al.* 2003) that was discussed earlier. Each agent may open one of two doors or listen. If either agent opens the door with the tiger behind it, a large penalty is given. If the other door is opened and the tiger door is not, a reward is given. If both agents open the same door a larger positive reward or a smaller penalty is given to reward this cooperation. If an agent listens, a small penalty is given and an observation is heard that is a noisy indication of which door the tiger is behind. While listening does not change the location of the tiger, opening a door causes the tiger to be placed behind each of the doors with equal probability.

The attributes for this problem were chosen as described in the previous section. Each agent remembers only the difference in the number of times an observation is heard up to some finite value k. This provides an approximation of the location of the tiger for each agent based on the observations that are heard. Recall that these estimates will differ for each agent due to the use of solely local information. The attribute values were set to be these observation differences with $k = 3$.

On this problem, our attribute-based method produces a drastically higher value than all the other algorithms we tested. This was accomplished in time similar to the fastest algorithm and with the second smallest controller. The BFS method quickly runs out of time and memory, resulting in a small controller with the second lowest value. DEC-BPI and the NLP method often settle for the safe option of listening for all steps and large negative rewards make higher valued solutions hard to find. Our approach narrows down the search to a small, but important set of attributes. The resulting optimal mapping consists of the very intuitive policy of an agent listening until it is reasonably sure it knows which door the tiger is behind and then opening the opposite door.

Meeting in a grid problems

The original meeting in a grid problem has 16 states, 5 actions and 2 observations and was introduced by Bernstein *et al.* In this problem, two agents must meet in a 2-by-2 grid with no obstacles. The agents begin diagonally across from each other and available actions are move left, right, up, or down and stay in place. Only walls to the left and right can be observed, resulting in each agent knowing only if it is on the left or right half. The agents cannot observe each other and do not interfere with other. Action transitions are noisy with the agent possibly moving in another direction or staying in place. A reward of 1 is given for each step that the agents share the same square, otherwise no reward is supplied.

We also examine versions of this problem in which the walls can be seen in four directions rather than just two. This causes the state to be fully locally observable by each agent in problem sizes up to three by three. Thus, while the other agent still cannot be observed, each agent knows where it is in the grid at all times. The problem sizes considered in this paper are 2-by-2 (16 states with 4 observations), 3-by-2 (36 states with 6 observations) and 3-by-3 (81 states with 9 observations). Note that these are much larger than the typical problems studied in DEC-POMDP literature.

For these problems, we chose the attribute values based

Two Agent Tiger Problem

BFS	DEC-BPI	NLP	Attribute Mapping
-14.115 (3 nodes, 12007s)	-52.633 (11 nodes, 102s)	-1.088 (19 nodes, 6173s)[1]	4.594 (7 nodes, 127s)

Meeting in a Grid Problems: original, 2-by-2, 3-by-2 and 3-by-3

BFS	DEC-BPI	NLP	Attribute Mapping
4.211 (2 nodes, 17s)	3.604 (7 nodes, 2227s)	5.658 (5 nodes, 117s)	6.285 (5 nodes, 7258s)
7.082 (2 nodes, 1669s)	3.794 (7 nodes, 1367s)	7.578 (4 nodes, 35s)	7.781 (4 nodes, 112s)
2.401 (1 node, 451s)	2.674 (6 nodes, 9827s)	6.228 (4 nodes, 160s)	6.910 (6 nodes, 7528s)
1.687 (1 node, 4763s)	2.675 (4 nodes, 13413s)	5.147 (3 nodes, 228s)	6.370 (5 nodes, 6169s)

Box Pushing Problem

BFS	DEC-BPI	NLP	Attribute Mapping
-2 (1 node, 1696s)	9.442 (3 nodes, 4094s)	54.230 (4 nodes, 1824s)[1]	158.492 (5 nodes, 4974s)

Table 1: The highest valued results by each method along with controller size and time in seconds. BFS and Attribute Mapping are optimal for the given conditions, while values for DEC-BPI and NLP represent means of 10 trials.

on the local states and state estimates for the agents. In the second and third problems (our 2-by-2 and 3-by-2 versions), we set the attributes to be the local states of each agent (their location in the grid). While this does not necessarily provide an optimal policy, remembering the current state provides a great deal of information while using a small amount of memory. In the first problem, because only walls to the left and right are observable to the agents, we included some of the agent's local state estimates as attribute values. For instance, if agent 1 successfully moves left on the first step, it knows that it is in the top left corner due to a left observation. Otherwise, the agent knows it is in one of the two right squares. Essentially, the state estimates allow the agents to take actions until they are reasonably sure they are in the top left square and then remain there. Depending on how sure we would like an agent to be, we can adjust the number of attribute values, and thus state estimates used. In the last problem (3-by-3), we reduced the number of attribute values to five as this produced the same mapping as using nine state attribute values. These values allow the agents to act based on if they in the top row, bottom row, or each of the three squares in the middle row.

The table lists results from the original meeting in a grid problem at the top, followed by our 2-by-2, 3-by-2, then 3-by-3 versions. As seen in the table, our approach outperforms all other algorithms that we tested on these problems. A small number of attributes permits high valued solutions that are in general much higher than those produced by BFS and DEC-BPI and noticeably higher than the NLP controllers. BFS is hindered by high resource requirements and this is especially noticeable as problem size grows and controller size is limited to one node. DEC-BPI consistently underperforms the NLP and attribute-based methods while requiring more time than other approaches in most cases. The NLP approach exhausts the given memory for larger controller sizes due to its own high resource requirements, but provides the second highest valued solutions in each problem in a very short amount of time. Our approach requires more time than some of the other methods, but this is because it provides an optimal action mapping for the given attributes.

Box pushing problem

A large domain was introduced by Seuken and Zilberstein (2007). This problem, with 100 states, 4 actions and 5 observations consists of two agents that can gain rewards by pushing different boxes. The agents begin facing each other in the bottom corners of a 4-by-3 grid and may turn right, turn left, move forward or stay in place. These movements are noisy and the two agents can never occupy the same square. The middle row of the grid contains one large box in the middle of two small boxes. The large box can only be moved by both agents pushing at the same time, while the small boxes can be moved by a single agent. The possible deterministic observations consist of seeing an empty space, a wall, the other agent, a small box or the large box. The upper row of the grid is considered the goal row and rewards are given when the boxes are pushed into this row. A reward of 100 is given if both agents push the large box and a reward of 10 is given for each small box that is pushed. A penalty of -5 is supplied for each agent that moves into a wall or the other agent and -0.1 is given on each step. Once a box is moved to the goal row, the environment resets to the original initial state.

Here, we chose the attribute values to be the problem observations. Thus, each agent only remembers the last observation seen, resulting in a reactive policy. The optimal mapping found by our approach consists of an agent moving forward when it sees a empty space or box and turning otherwise. This results in a high valued policy that can often receive the large reward in a few steps. The values of the plans generated by the other techniques are significantly lower. BFS cannot solve a controller that is large enough to to produce a non-trivial policy, and DEC-BPI and the NLP method get stuck at relatively low quality solutions. While

[1]These results utilize controllers with fixed actions at each node.

the time used by our method is slightly higher than the other methods, we believe this is compensated for by the large increase in value.

Conclusion

In this paper, we introduced a simple way to overcome the high memory requirements that often hinder the generation of plans for DEC-POMDPs by remembering a limited number of important characteristics of an agent's action and observation history. To accomplish this, a combination of automatic and user defined attributes are used to produce a reduced complexity model which represents a finite-state controller with a well-motivated structure. The solution of this model provides a policy for the given problem that is not only high valued but also compact and easily readable.

We also present a branch and bound algorithm that provides an optimal action choice for these attributes. This algorithm allows the action mappings to be found in a scalable way. Additionally, our experiments showed that when compared to the current state-of-the-art infinite-horizon DEC-POMDP approximate algorithms, our attribute-based representation produces significantly higher valued plans. These results demonstrate that our method can scale to larger problems and provide better plans for a range of DEC-POMDP problems.

In the future, we plan to pursue several interesting extensions and open questions. These include incorporating commitments between agents and using factored controllers to improve computational efficiency when agents each have various multi-valued attributes. We would also like to determine when attributes can be chosen in such a way that an optimal mapping is also an optimal policy for the problem.

Acknowledgements This work was supported in part by the Air Force Office of Scientific Research (Grants No. FA9550-05-1-0254 and FA9550-08-1-0181) and by the National Science Foundation (Grant No. 0535061). Any opinions, findings, conclusions or recommendations expressed in this manuscript are those of the authors and do not reflect the views of the US government.

References

Christopher Amato, Daniel S. Bernstein, and Shlomo Zilberstein. Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs. Technical Report CS-07-70, University of Massachusetts, Department of Computer Science, Amherst, MA, 2007.

Christopher Amato, Daniel S. Bernstein, and Shlomo Zilberstein. Optimizing memory-bounded controllers for decentralized POMDPs. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, Vancouver, Canada, 2007.

Raphen Becker, Shlomo Zilberstein, Victor Lesser, and Claudia V. Goldman. Solving transition-independent decentralized Markov decision processes. *Journal of AI Research*, 22:423–455, 2004.

Daniel S. Bernstein, Shlomo Zilberstein, and Neil Immerman. The complexity of decentralized control of Markov decision processes. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, 2000.

Daniel S. Bernstein, Eric Hansen, and Shlomo Zilberstein. Bounded policy iteration for decentralized POMDPs. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 1287–1292, Edinburgh, Scotland, 2005.

Daniel S. Bernstein. *Complexity Analysis and Optimal Algorithms for Decentralized Decision Making*. PhD thesis, University of Massachusetts, Amherst, MA, 2005.

Alan Carlin and Shlomo Zilberstein. Value-based observation compression for DEC-POMDPs. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems*, Estoril, Portugal, 2008.

Rosemary Emery-Montemerlo, Geoff Gordon, Jeff Schneider, and Sebastian Thrun. Approximate solutions for partially observable stochastic games with common payoffs. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 136–143, New York, NY, 2004.

Eric A. Hansen, Daniel S. Bernstein, and Shlomo Zilberstein. Dynamic programming for partially observable stochastic games. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, pages 709–715, San Jose, CA, 2004.

Ranjit Nair, David Pynadath, Makoto Yokoo, Milind Tambe, and Stacy Marsella. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 705–711, Acapulco, Mexico, 2003.

Frans Oliehoek, Matthijs Spaan, Shimon Whiteson, and Nikos Vlassis. Exploiting locality of interaction in factored Dec-POMDPs. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, Estoril, Portugal, 2008.

Marek Petrik and Shlomo Zilberstein. Average-reward decentralized markov decision processes. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, pages 1997–2002, Hyderabad, India, 2007.

Sven Seuken and Shlomo Zilberstein. Improved memory-bounded dynamic programming for decentralized POMDPs. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, Vancouver, Canada, 2007.

Daniel Szer and Francois Charpillet. An optimal best-first search algorithm for solving infinite horizon DEC-POMDPs. In *Proceedings of the Sixteenth European Conference on Machine Learning*, Porto, Portugal, 2005.

Daniel Szer, Francois Charpillet, and Shlomo Zilberstein. MAA*: A heuristic search algorithm for solving decentralized POMDPs. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, Edinburgh, Scotland, 2005.