Achieving Goals in Decentralized POMDPs

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

ABSTRACT

Coordination of multiple agents under uncertainty in the decentralized POMDP model is known to be NEXP-complete, even when the agents have a joint set of goals. Nevertheless, we show that the existence of goals can help develop effective planning algorithms. We examine an approach to model these problems as indefinite-horizon decentralized POMDPs, suitable for many practical problems that terminate after some unspecified number of steps. Our algorithm for solving these problems is optimal under some common assumptions – that terminal actions exist for each agent and rewards for non-terminal actions are negative. We also propose an infinite-horizon approximation method that allows us to relax these assumptions while maintaining goal conditions. An optimality bound is developed for this sample-based approach and experimental results show that it is able to exploit the goal structure effectively. Compared with the state-of-the-art, our approach can solve larger problems and produce significantly better solutions.

1. INTRODUCTION

The decentralized partially observable Markov decision process (DEC-POMDP) is a powerful and attractive way to model cooperative sequential multiagent decision making under uncertainty. It is an extension of the partially observable Markov decision process (POMDP), allowing multiple agents to choose their individual actions based on their individual observations and jointly affect a global state and reward functions. In addition to the uncertainty about the global state, each agent must cope with imperfect information about the knowledge and actions of the other agents. As this is a cooperative model, solutions seek to maximize a shared objective function using solely local information to determine each agent's action.

Many DEC-POMDP domains involve a set of agents completing some task or achieving a goal. The number of steps needed to achieve the goal is unknown, but the problem ends once it has been completed. Examples of these domains include agents targeting and catching an object, meeting in an environment, cooperatively exploring and experimenting in an environment, or moving an object or sets of objects. In each case, agents have to cooperate while using only local information to finish the task in an efficient manner. For instance, meeting in an environment requires the agents to choose a central location and each determine a set of paths that will allow the location to be reached in the quickest manner while also considering what paths the other agents may take. This is a general class of problems that has many natural real world applications.

Several optimal and approximate algorithms for solving DEC-POMDPs have been proposed. Goldman and Zilberstein [5] discuss the complexity of a type of goal-oriented problems under different assumptions, but do so in a finite-horizon context in which there is a fixed number of steps until completion. For general finite-horizon problems, Hansen et al. [7] and Szer et al. [15] present optimal algorithms, while others provide approximate methods [9, 12]. Approximate algorithms for infinite-horizon problems have been proposed as well [1, 3, 14].

In goal based problems, the number of steps until completion depends on the actions that are taken and not a fixed number of steps as in finite-horizon problems. Solving these problems as infinite-horizon essentially involves solving the same problem over and over again. Also, infinite-horizon solutions are susceptible to changes in the discount factor, which maintains a bounded value. We can instead model these problems as indefinite-horizon, which terminate after some unknown number of steps. We base our indefinite-horizon representation on the action-based termination model used in POMDPs [6]. Our model assumes that terminal actions can be taken at any step and cause the problem to stop. In order to guarantee that problems terminate after a finite number of steps and optimal solutions can be found, we must also assume that all rewards are negative except for those generated from the terminal actions. Under these assumptions, we provide an optimal dynamic programming algorithm for indefinite-horizon DEC-POMDPs with action-based termination.

We can relax some of the assumptions made for action-based termination to allow us to model a wider range of problems. While we can no longer guarantee that these problems will terminate after a finite number of steps, the presence of goals allows us to implement a sampling approach that can take advantage of the added structure to produce high quality infinite-horizon solutions. We present an approximate algorithm for solving a class of *goal-directed* DEC-POMDPs, which have a well defined termination condition and any type of reward functions. We show that this sampling approach can significantly outperform the state-of-the-art approximate infinite-horizon DEC-POMDP algo-

Cite as: Title, Author(s), *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. XXX-XXX.

rithms on a number of goal-directed problems.

The rest of the paper is organized as follows. We first describe the DEC-POMDP model, its solution and the relevant previous work. We then present our model for indefinite-horizon DEC-POMDPs with action-based termination and a dynamic programming algorithm to determine optimal solutions. We then describe the more general goal-directed model and an approximate algorithm that is able to take advantage of goals in DEC-POMDPs. We provide a bound on the likelihood that an *epsilon*-optimal solution is found and show experimentally that this approach can produce high quality solutions to a range of domains.

2. BACKGROUND

We begin by reviewing the DEC-POMDP framework as well as how to represent finite and infinite-horizon solutions. We also review previous work with goals in DEC-POMDPs as well as the relevant optimal and approximate algorithms.

2.1 The DEC-POMDP model

A DEC-POMDP can be defined $\langle I, S, \{A_i\}, P, R, \{\Omega_i\}, O\rangle$ with I, a finite set of agents, S, a finite set of states with designated initial state distribution b_0, 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 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 seeing the set of observations $\vec{o}$ given the set of actions $\vec{a}$ was taken which results in state s' and T, a horizon or finite number of steps after which the problem terminates.

At each step, every agent chooses an action based on their local observation histories, resulting in an immediate reward and an observation for each agent. Note that because the state is not directly observed, it may be beneficial for the agent to remember its 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 local policies, one for each agent in the problem. The goal is to maximize the total cumulative reward until the horizon is reached, beginning at some initial distribution over states. In the infinite-horizon problem, T is infinity and the decision making process unfolds over an infinite sequence of steps. In order to maintain a finite sum over the infinite-horizon, we employ a discount factor, $0 \le \gamma < 1$.

2.2 Policy representations

To represent policies for finite-horizon DEC-POMDPs, policy trees can be used. These are the same as those employed for POMDPs except there is a now policy tree for each agent. An agent's policy tree can be defined recursively. The tree begins with an action at the root and a subtree is defined for each observation that the agent could see. This continues until the horizon of the problem is achieved at the leaf nodes. Thus, the agent's choice of actions is defined by a path through the tree that depends on the observations that it sees. The joint policy is defined by a set of policy trees, one for each agent and can be evaluated by summing the rewards at each step weighted by the likelihood of transitioning to a given state and observing a given set of observations.

For infinite-horizon DEC-POMDPs, finite state controllers can be used instead of policy trees. While infinite size controllers may be needed to represent optimal policies, finite state controllers present an appealing way to model infinite-horizon DEC-POMDP policies with finite memory. Each agent's policy can be represented as a local controller and the resulting set of controllers supply the joint policy. Because fixed memory is being used, it can be beneficial for agents to use stochastic controllers. Each finite state controller can formally be defined by the tuple $\langle Q, \psi, \eta\rangle$, where Q is the finite set of controller nodes. The parameter $\psi : Q \to \Delta A$ is the action selection model for each node, which defines a probability that an action will be chosen in a given node. The parameter $\eta : Q \times A \times O \to \Delta Q$ represents the node transition model for each node. This provides the probability of transitioning from one node to another after an action has been taken and an observation has been seen. For n agents, the value for being at nodes $\vec{q}$ and state s is

$$V(\vec{q},s) = \sum_{\vec{a}} \prod_i^n P(a_i|q_i) \Big[R(s,\vec{a}) + \gamma \sum_{s'} P(s'|\vec{a},s) \cdot \sum_{\vec{o}} O(\vec{o}|s',\vec{a}) \sum_{\vec{q'}} \prod_i^n P(q_i'|q_i,a_i,o_i) V(\vec{q'},s') \Big]$$

Note that the values for either policy representation can be calculated offline in order to determine trees or controllers that can then be executed online for distributed control.

2.3 Previous work

Goldman and Zilberstein [5] modeled goal oriented DEC-POMDPs as finite-horizon problems with a set of goal states and negative rewards in all non-goal states. Adding these goal states to the finite-horizon problem was shown to not change the complexity of general DEC-POMDPs (NEXP-complete [4]) and independent transition and observation (IT-IO) DEC-MDPs (NP-complete [2]) when no additional assumptions were made. In IT-IO DEC-MDPs, Goldman and Zilberstein also assume that each agent also has access to a no-op action in goal states, where the agent essentially waits for the other agents to reach the goal. When a unique goal state was assumed or when a single goal is always known to have higher value than the others, the IT-IO case was shown to be P-complete.

To optimally solve a general finite-horizon DEC-POMDP, Hansen et al.'s dynamic programming algorithm [7] can be used. While this algorithm solves partially observable stochastic games, DEC-POMDPs are a subclass in which the agents share a common payoff function. In this algorithm, a set of k-step policy trees, one for each agent, is generated from the bottom up. That is, on the kth step of the problem, each agent will perform just a single action (a 1-step policy tree), so we begin the building of policy trees with considering taking any action on the last step. We can then remove any action that is not useful no matter what problem state we are at or what possible actions the other agents take. That is, if a tree t has lower value than another tree, $\hat{t}$ for all states and trees of the other agents, it is removed, or pruned, because it is always better to use $\hat{t}$. This pruning can be done for all agents until trees can no longer be pruned.

On the next step of the algorithm, we generate all 2-step policy trees. This is done for each agent by considering any root action and for any observation that is seen, choosing some 1-step tree. If an agent has $|T|$ 1-step trees, $|A|$ actions, and $|\Omega|$ observations, there will be $|A||T|^{|\Omega|}$ 2-step trees. After this exhaustive generation of next step trees is completed for each agent, these sets can then be pruned to reduce their number. This generation and pruning continues until the given horizon is reached.

Problems that repeat or continue in some way after goals have been reached can be solved using infinite-horizon algorithms. Three approximate approached have been developed by Bernstein et al. [3], Amato et al. [1] and Szer and Charpillet [14] which optimize fixed size finite state controllers. Bernstein et al. and Amato et al. use linear programming and nonlinear programming (NLP) respectively to find parameters for stochastic controllers. Szer and Charpillet use best-first search to construct deterministic controllers.

3. INDEFINITE-HORIZON DEC-POMDPS

In this section, we first discuss the indefinite-horizon model as well as indefinite-horizon approaches for POMDPs. We then provide the DEC-POMDP model and an algorithm to solve it, including a proof that this approach will provide an optimal indefinite-horizon solution.

3.1 Overview

In many benchmark and real world problems, a set of agents must achieve some goal in order to receive maximum value. Benchmark problems include the two agent tiger problem [9], the box pushing problem [12] and the meeting in a grid problem [3]. After this goal has been reached, the problem either resets (tiger and box problems) or the problem effectively stops (grid problem). These problems can be modeled as a type of indefinite-horizon problem in which the problem stops after an unknown number of steps after some goal condition has been reached. This is a natural and expressive class of problems in which a set of agents must complete some task such as navigate to a goal location or simultaneously or independently choose a goal action.

Goldman and Zilberstein discussed goal-oriented problems, but modeled them as finite-horizon and required the presence of no-op actions when agents' transitions are independent. Other approaches have ignored the presence of a goal and have used general infinite-horizon DEC-POMDP algorithms to find solutions. This causes the problem to be solved over and over again and the solution to be dependent on the horizon or discount chosen. The alternative is to model these problems as indefinite-horizon, which removes the need to either know the proper horizon beforehand or choose an often meaningless discount factor.

Indefinite-horizon models for POMDPs have been proposed by Patek [10] and Hansen [6]. Patek describes *partially observed stochastic shortest path problems*, which make the following assumptions: (1) existence of a policy that terminates with probability one, (2) any policy that does not guarantee termination has infinite negative reward and (3) termination is fully observable. He proves that there is a stationary optimal policy, and in the limit value iteration and policy iteration will converge to such a policy. Hansen proposes the *action-based termination* model which, in addition to Patek's assumptions also assumes (4) a set of terminal actions, (5) all non-terminal actions have negative reward, and (6) terminal actions can have a positive or negative reward that depends on the state. The action-based termination model allows simpler algorithms as proper policies, which guarantee termination, are easily found. This greatly simplifies the use of value and policy iteration with the model.

3.2 Indefinite-horizon framework

Hansen's action-based termination model can be extended to DEC-POMDPs with a minor change. We will require that each agent has a set of terminal actions and the problem stops when one of these is taken by each agent simultaneously. We also require negative rewards for non-terminal actions which implies the other assumptions made by Patek and Hansen. We could have assumed the problem ends when any agent chooses a terminal action without adding much complication to the algorithm and analysis. We could have also used a set of fully observable goal states that the agents must reach, but this complicates matters as an agent's actions depend on only local information, but reaching the goal may depend on certain information being observed by all agents. Thus, even if the goal state is completely observable, estimation of the intermediate states may not possible with local information, making transitions to the goal unlikely or impossible. Therefore, it is an open question whether it is decidable to find an ϵ-optimal indefinite-horizon DEC-POMDP policy without assumptions that are more restrictive than those made by Patek.

We can define an indefinite-horizon DEC-POMDP with action-based termination in the same way as a finite-horizon DEC-POMDP in that a discount factor is not used, but with the added difference that there is no specified horizon. Instead, the problem stops when a terminal action is chosen by each agent. Assumptions (1)-(6) then allow us to describe a dynamic programming algorithm that returns an optimal solution for indefinite-horizon DEC-POMDPs in a finite number of steps. This algorithm is an extension of the indefinite-horizon POMDP algorithm presented by Hansen [6] and the optimal finite-horizon DEC-POMDP algorithm presented by Hansen et al. [7]. We can build policy trees for each agent by first considering only terminal actions on the last step and building up the tree by adding only non-terminal actions on each successive step. This process of generating next step trees and pruning continues until we can be sure that the optimal set of trees has been produced.

Unfortunately, the decentralized case does not allow an error bound to be calculated in the form of a Bellman residual as is used by the POMDP method. The Bellman residual is found from the maximum difference in value for all belief states, but in DEC-POMDPs, the value of an agent's policy depends not only on the state of the system, but also on the policies chosen by the other agents. Thus, constructing a Bellman residual in the DEC-POMDP case is an open question. Instead we can calculate an upper bound on the horizon of any optimal policy and then generate all trees up to that bound using dynamic programming. A set of policy trees that provides the highest value for the given initial state distribution, regardless of horizon, is an optimal solution for the problem.

We first show how to calculate the upper bound on the horizon of an optimal policy.

LEMMA 3.1. *An optimal set of indefinite-horizon policy trees must have horizon less than $k_{max} = (R_{now} - R^T_{max})/R^{NT}_{max}$ where R^T_{max} is the value of the best combination of terminal actions, R^{NT}_{max} the value of best combination of non-terminal actions and R_{now} is the maximum value attained by choosing a set of terminal actions on the first step given the initial state distribution.*

PROOF. This proof centers on the fact that the reward for all non-terminal actions is negative. After k steps of dynamic programming, the maximum value attained by any set of policies is $(k-1)R^{NT}_{max} + R^{T}_{max}$. If this value is less than the best value of immediately choosing terminal actions at the initial state distribution, R_{now}, then the agents are better off ending the problem on the first step. Thus, when $(k-1)R^{NT}_{max} + R^{T}_{max} \leq R_{now}$ we know that it is more beneficial to choose some combination of terminal actions at an earlier step. Any set of trees with horizon higher than k will have even lower value, so $k_{max} = (R_{now} - R^{T}_{max})/R^{NT}_{max}$ $\square$

This allows us to bound the horizon, and then we can just choose the set of trees for any horizon up to k_{max} that provides the highest value for the initial state distribution.

Theorem 3.2. *Our dynamic programming algorithm for indefinite-horizon POMDPs returns an optimal set of policy trees for the given initial state distribution.*

Proof. This follows almost directly from Lemma 3.1. Our dynamic programming algorithm solves the indefinite-horizon problem up to horizon k_{max}, which we have shown represents an upper bound on the horizon of optimal policies. Because our algorithm represents an exhaustive search (except for pruning) in policy space up to horizon k_{max}, an optimal optimal set of policy trees must be contained in the resulting sets of trees. Pruning does not remove any policy trees that will be useful in any optimal policy (because it is only done when there is another tree that is always better for all states and trees or subtrees of the other agents, as shown in [7]). A set of trees that provides the highest value for the given initial state then represents an optimal policy for the indefinite-horizon DEC-POMDP. $\square$

If we choose R_{now} to be the maximum value attained by choosing a set of terminal actions for any possible initial state distribution, the resulting sets of policy trees will include an optimal set for any initial distribution. Also, if we allow any agent to terminate the problem, we must include both terminal and non-terminal actions on the first step of the algorithm and then ensure that at least one agent chooses a terminal action on the last step of the problem.

Many DEC-POMDP domains can be naturally modeled as indefinite-horizon with action-based termination. One example is the search for a moving target problems discussed by Hansen. When adapted to the decentralized case, either both agents must both simultaneously "catch" the target or one *is* the target and the goal is to meet by using local information. The later case is exactly modeled by the meeting in a grid benchmark DEC-POMDP problem [3]. The terminal actions for this grid problem consist of the agents deciding to stay in the same place when they believe they are in the same grid square as the other agents. The multiagent tiger benchmark problem [9] in which the problem resets when any agent chooses a door that it believes the tiger is behind is also naturally captured by this model. The terminal actions for the tiger problem are choosing to open a door. Many other robot navigation and task oriented problems can be similarly represented using action-based termination.

4. GOAL-DIRECTED DEC-POMDPS

If we relax assumptions (1)-(6) given above, but retain the notion of a goal, we may no longer be able to provide an optimal solution that ends after a bounded number of steps. Instead, we have a class of general infinite-horizon DEC-POMDPs that has some structure which can allow us to produce high quality approximate solutions by focusing on policies that achieve the goal. In this section, we first give an overview of this class of DEC-POMDPs and then provide a sample-based algorithm for generating solutions.

Algorithm 1: Goal-directed controller generation

```
input  : The total number of samples desired, $n_{total}$,
         and the number of those to retain, $n_{best}$
output: A set of controllers for the agents, $\hat{Q}$
begin
    $trajectories \leftarrow getTrajectory(b_0, goal, n_{total})$
    $bestTraj \leftarrow chooseBest(n_{best}, trajectories)$
    for each agent, $i \in I$ do
        $Q_i \leftarrow buildController(bestTraj_i)$
        $\hat{Q}_i \leftarrow reduceController(Q_i)$
    $change \leftarrow true$
    while change do
        $change \leftarrow findBestActions(\hat{Q})$
        $change \leftarrow findTransitions(\hat{Q})$
    return $\hat{Q}$
end
```

4.1 Overview

We will discuss a class of infinite-horizon DEC-POMDPs with specified goal criteria. We call these problems *goal-directed* and require that the problem terminates or resets if any of the following hold (1) the set of agents reach a global goal state, (2) a single agent or set of agents reach local goal states, or (3) any chosen combination of actions and observations is taken or seen by the set of agents.

These assumptions permit a very general class of problems that is common in benchmark domains and real world applications. Because the two agent tiger [9] and meeting in a grid problems [3] fit in the action-based termination model, they also fit here, but the box pushing problem [12] is also included in this class. The box pushing problem resets when at least one agent pushes a small box or when both agents push a large box into a goal row. Thus, there are multiple goal states that can be achieved by the agents, but no terminal actions.

Real world problems include those discussed for indefinite-horizon DEC-POMDPs as well as those with more general goals and reward structure. For instance, consider a set of agents that must perform different experiments at certain research sites. Some of these sites may require multiple agents performing some experiment together in order to get the most scientific value, while other sites may require a specific tool be used by a single agent. Positive rewards are given for successfully performing experiments at each site and the task is completed when all sites have been experimented on. This is a version of the Mars rover problem discussed in [2]. Many other problems have these characteristics in which the agents must complete some task with clear termination conditions.

4.2 Algorithmic approach

In order to solve these problems, we develop a sampling approach for DEC-POMDPs. Sampling has been effective in reducing the necessary search space in many planning problems. For instance, in POMDPs, work has primarily dealt with sampling belief states and generating policies for these sampled states. Methods have been developed which can bound the sample size necessary for determining ϵ-optimal policies [11] and empirically, very high quality policies can be found with a much smaller set of belief states [11, 13]. Unfortunately, it is not straightforward to extend these methods to DEC-POMDPs. This is due to the fact that a shared belief state can no longer be calculated and policies for each agent must be evaluated in the context of policies for all other agents and executed in a decentralized manner.

In our approach, we use the presence of goals to construct an algorithm which samples the policies of the agents and produces an infinite-horizon solution. The policies for all agents are sampled simultaneously by using trajectories which begin at the initial state and terminate when the goal conditions are met. Because planning is conducted in a centralized fashion, it is always known when the goal is reached. If the goal can always be achieved, the goal conditions focus and limit the possible trajectories and with sufficient samples we could construct all possible policies for the set of agents. This would result in an exhaustive policy search and an optimal joint policy could then be chosen from the set of possibilities.

When a subset of the possible trajectories is generated, a partial policy search is conducted and we must decide what is done when observations are seen that are not observed in the trajectories. Also, due to decentralized execution of policies, even if the goal is reached during the trajectories, if an unseen observation occurs it could cause the agents to fail to reach the goal. Because it may not be possible to ensure the goal is reached, we must ensure a valid policy is generated for any number of steps. For these reasons, we choose to generate finite state controllers from the sampled trajectories. The controllers are optimized to provide policy choices for observations that are and are not part of the trajectories and action choices for histories of any possible length. The controllers will also allow the best decentralized policies to be chosen while considering the possible choices of the other agents.

4.2.1 *Algorithm description*

Our approach is given in Algorithm 1. First, we generate a set of trajectories which consist of action and observation sequences for each agent that begin at the initial state distribution and end when the goal is achieved. Actions are chosen randomly by each agent and observations are chosen based on their likelihood after the set of actions has been taken. Trajectories that have not reached the goal after a given cutoff are discarded.

The trajectories generate rewards for the set of agents at each step until the goal is reached. These can be summed by $R(b_0, \vec{a}_0) + \ldots + \gamma^T R(b_{goal}, \vec{a}_{goal})$ where the reward is weighted by the state probabilities $R(b, \vec{a}) = \sum_s b(s)R(s, \vec{a})$ and T is the number of steps used to reach the goal. A heuristic that considers the value of trajectories can be used to focus and reduce the number considered. This is done by choosing some number of trajectories that produce the best value and retaining only them. While some low value trajectories may be valuable to determine what actions should be taken in certain cases (such as when actions fail to have the desired outcome), in general, the highest valued trajectories are the most useful for generating high valued policies.

A controller for each agent can then be generated from these trajectories. This is accomplished by first creating a tree whose root node represents the initial state and leaves represent the goal being achieved. Starting from the root, new nodes are created for each action that was taken at the first step of the trajectories. From these nodes, new nodes are then created for each observation that was seen after the given action was taken. This process continues until the goal has been reached, resulting in a tree which represents any action and observation sequence that is present in any of that agent's trajectories. An example of a set of trajectories and an initial controller is shown in Figure 1. It is for the possible policies that are generated at this step for which we derive a bound in our analysis.

To better use trajectories and representation space, this tree can then transformed into a controller and then optimized. A controller is constructed by adding transitions after the goal has been reached into an absorbing node or back to the root node, depending on the problem dynamics. To ensure that an action is defined for any possible action and observation history we can arbitrarily assign controller transitions for observations that are not a part of any trajectory. For instance, the controller can transition back to the beginning or stay in place for unseen observations. The transitions can be improved during the optimization phase.

Because these controllers are generated from trees, as soon as two trajectories differ, the controller branches out and a separate path is created until the goal is reached. This is true even if the trajectories become the same for later sequences. For this reason, we can reduce the size of the controllers by starting from the last node of each trajectory and determining if the same actions are possible, the same observations are seen and the same node is transitioned to. If this is the case, the nodes can be merged. This can greatly reduce the size of the controllers without reducing their expressiveness.

We then need to choose the best of the possible actions at each node of the controller. This can be done by any search method such as branch and bound. The different combinations of action choices are evaluated for the set of agents and the one with the highest value is chosen. Determining the actions for each agent's controller requires at most $\prod_{i,q_i} |A_i^{q_i}|$ steps for each agent i where $|A_i^{q_i}|$ represents the number of possible actions for node q_i. In many cases, we can stop after this search and have a high quality infinite-horizon policy for each agent, but it may also be beneficial to adjust the controller transitions.

A more compact or higher valued controller may be able to be constructed if the tree-like structure is adjusted and transitions between different nodes are used. Because it is often intractable to search through all possible controller transitions for all agents, we use a heuristic method that allows transitions to be changed within a controller. This approach tests each node and observation to determine if a higher value can be attained by transitioning to a different resulting node while keeping other transitions in the agent's controller and all other agents' controllers fixed. If an improvement is found, the transition is updated for that observation. We can continue optimizing the action choices and transition choices for the agents until no further improvements can be made.

4.2.2 *Example*

An example of policy generation for an agent is shown

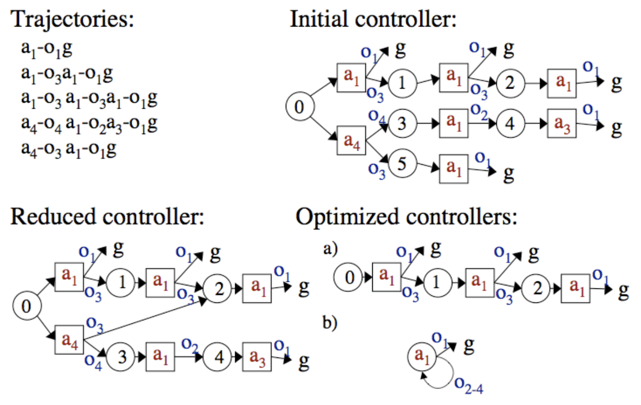

Figure 1: An example of a controller generated from a set of trajectories. The trajectories all begin at an initial state and end when a goal condition has been satisfied. The initial controller is then created from these trajectories and redundant nodes are combined to produce the reduced controller. This controller is then optimized by evaluating it in conjunction with the controllers of the other agents and optimizing the action choices and transition choices as shown in a) and b) respectively.

in Figure 1. This example is motivated by a version of the meeting in a grid problem in which the agents must navigate to a certain grid location while movements can slip, resulting in the agent staying in place. We begin with a set of action and observation sequences which represent trajectories starting at an initial state and end when the goal condition is satisfied (labeled g throughout). These trajectories represent different paths that the agent took to achieve the goal. While it was possible that the agent reached the goal location after taking one action, because movements are noisy and multiple paths exist, other instances required multiple iterations of that action or a different set of actions.

The trajectories are then used to generate an initial controller. For each trajectory, there is a path in the controller that reaches the goal. Circles are used to represent nodes, squares represent action choices and observations are labeled above the given transition arrow. This initial controller is a tree, except we can consider transitioning to any node once the goal has been reached. We can then reduce the controller by noticing that node 5 and node 2 are equivalent. That is, each takes only action a_1 and after observing o_1 transitions to the goal. Thus, the two nodes can be merged, reducing the controller size from 6 to 5.

The optimization phase further reduces the controller size by choosing actions and adjusting transitions. The action choices that provide the highest value are chosen in part a, producing a 3 node controller with fixed actions. The transitions after each observation, whether part of the trajectories or not, are then optimized, resulting in a single node that chooses action a_1 until the goal is reached. The optimized controller is very simple and provides a high valued policy for this version of the meeting in a grid problem.

4.3 Analysis

The initial policy for the controller is built from sampled trajectories of the domain. With an unlimited number of samples and a bound on the horizon, by choosing the best actions, the initial controllers would be optimal. More interestingly, it is possible to bound the error of the controllers obtained, based on the number of samples used to generate them. We show that bounds developed in the context of POMDPs [8] may be adapted to our setting. The goal is to show that with probability at least $1-\delta$ we can construct controllers that have value within ϵ of the optimal set of controllers.

These bounds are based on an analysis similar to that of learning-theoretic bounds, with a few modifications. In machine learning, typically all samples may be used in evaluating every hypothesis. The bounds then rely on the large number of samples available for testing each hypothesis. This is not the case in our setting. When evaluating a deterministic joint policy π, only trajectories with the same actions as π for the observations that are seen may be used. Also, as mentioned above, because policies are produced from the trajectories, actions will not be specified for all observation sequences. This does not present a problem in our analysis, since the set of samples used to define the trajectories is identical to the set of samples used to define the policies.

We adopt the analysis from [8], section 5.1, to derive the bounds. While this analysis produces somewhat loose bounds, it lays the foundation for more sophisticated bounds, which remains the subject of future work. We use $\pi \in \Pi$ to denote a joint deterministic policy from the set of all possible policies. We also use A_i to denote the set of available actions for agent i, and $A = \prod_i^n A_i$ with $|A| \geq 2$. We assume an upper bound on the number of steps required to reach the goal, denoted by T. The set of available trajectories is $\mathcal{H}$, assuming that the actions taken are chosen randomly with a uniform probability, and independently in each step. Finally, let $V_{\max}$ be the maximal difference between the values of the policies in Π. Using this notation, we show that if the optimal policy is π^*, the policy obtained from the samples is $\tilde{\pi}(\mathcal{H})$ and given a sufficient number of samples, then for any ϵ we can bound:

$$\mathrm{P}\left[V^{\pi^*}(s_0) - V^{\tilde{\pi}(\mathcal{H})}\right] \geq \epsilon$$

The following lemma describes the probability of generating a trajectory, h, that is consistent with any joint policy π. We use $acc_\pi(h)$ to denote the presence or absence of this consistency. For lack of space we only state lemmas that are different from [8]. We provide the corresponding lemma or theorem numbers from this reference below. The proofs are straightforward and thus omitted.

Lemma 4.1 (An adaptation of Lemma 5.1). *For any joint deterministic policy:*

$$\mathrm{P}\left[acc_\pi(h) = 1\right] = \frac{1}{|\mathcal{A}|^T}$$

We can then bound the probability that there is a joint policy that is consistent with less than $|\mathcal{H}|/|\mathcal{A}|^{T+1}$ samples.

Lemma 4.2 (An adaptation of Lemma 5.3). *Assuming $|\mathcal{H}| > |\mathcal{A}|^{T+3}\log(2|\Pi|/\delta)$, the probability that there exists a policy $\pi \in \Pi$ for which $|\{h|acc_\pi(h) = 1\}| < |\mathcal{H}|/|\mathcal{A}|^{T+1}$ is at most $\delta/2$.*

Together with Lemma 5.4, the theorem below follows.

Theorem 4.3 (An adaptation of Theorem 5.5). *Assuming $|\mathcal{H}| > |\mathcal{A}|^{T+3}(V_{\max}/\epsilon)^2\log(2|\Pi|/\delta)$, we have with* *probability at least $1-\delta$ simultaneously for all $\pi \in \Pi$ that:*

$$\left| V^{\pi}(s_0) - \tilde{V}^{\pi}(s_0) \right| \leq \epsilon,$$

where $\tilde{V}$ is the value obtained from the samples.

The theorem above shows that the value of the generated policy is with probability $1-\delta$ at most ϵ from the true value of the policy. To bound the difference between the generated policy $\tilde{\pi}$ and the optimal policy π^*, we can use the triangle inequality and the optimality of $\tilde{\pi}$ on the samples to prove the following.

COROLLARY 4.4. *From Theorem 4.3 we have:*

$$\mathrm{P}\left[V^{\pi^*}(s_0) - V^{\tilde{\pi}(\mathcal{H})} \geq \epsilon \right] \leq \delta.$$

5. EXPERIMENTS

In this section, we evaluate the performance of our goal-directed approach as well as three state-of-the-art infinite-horizon DEC-POMDP approximation algorithms. We compare our approach to the NLP method of Amato et al. [1], Bernstein et al.'s DEC-BPI method [3] and Szer and Charpillet's BFS algorithm [14]. Results are given for three common benchmark domains as well as a much larger scientific coordination task that was mentioned previously.

5.1 Experimental setup

For the NLP and DEC-BPI approaches, each algorithm was run until convergence was achieved with ten different random deterministic initial controllers, and the mean values and running times were found for a range of controller sizes. The nonlinear program solver snopt on the NEOS server was used to determine solutions for the NLP approach. The algorithms were run on increasingly larger controllers until memory was exhausted (2GB) or time expired (4 hours). We then report the results for the controller size with the highest value for each method.

For our goal-directed approach, our algorithm was also run until it converged ten times and the mean values and running times are reported. We also provide the sizes of typical controllers that are generated. The total number of trajectories generated and retained were 5000000 and 25 for the two agent tiger problem, 1000000 and 10 for the meeting in a grid and box pushing problems and 500000 and 5 for the two rover problems. It is worth noting that due to the small number of samples retained, our bound on the optimality of the solution does not hold in the results below. Sampling time ranged from less than 30 seconds for the tiger problem to approximately 5 minutes for the stochastic version of the rover problem.

The BFS algorithm was run until on increasing controller sizes until memory or time was exhausted. The solution reported represents the optimal deterministic controller for the largest solvable controller size. 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.

5.2 Domain descriptions

We test the algorithms on three benchmark domains and two versions of a very large rover coordination problem. The benchmark domains that were used are the two agent tiger problem [9], the meeting in a grid problem [3] and the box pushing domain [12]. These problems were discussed earlier and for more details see the references. On all of the test domains a discount factor of 0.9 was used.

The new domain that we propose has more than twice the number of states of the previous largest domain. This problem has 256 states, 6 actions and 8 observations and involves two rovers performing certain scientific experiments at a set of sites. The agents can conduct scientific experiments at four possible sites by choosing to drill or sample. These sites are arranged in a two-by-two grid in which the rovers can independently move north, south, east and west. Two of the sites require only one agent to sample them while two of the sites require both agents to drill at the same time in order to get the maximum reward. If a site only needs to be sampled, but it is drilled instead, the site is considered ruined and thus a large negative reward is given. If a sites requires drilling, but is sampled instead, a small positive reward is given. Once a site is sampled or drilled any further experiment is considered redundant and incurs a small negative reward. The rovers can fully observe their own local location as well as whether an experiment has already been performed at that site. When at least one experiment is performed at each site, the problem is reset. We provide results for a version of this problems in which the rovers can deterministically move in the intended direction and a version in which there is a small probability that movement fails and the rover stays in place.

5.3 Results

The experimental results are given in Table 1. In all domains except the meeting in a grid problem, significantly higher values are found with our goal-directed approach. The BFS approach is limited to very small controllers so even though an optimal deterministic controller is generated for the given controller sizes, these controllers were not large enough to produce a high value. In the rover problems, it could not find a solution for a one node controller before time was exhausted. DEC-BPI can solve larger controllers, but has a high time requirement and generally performs poorly in these domains. The NLP method performs better, producing the highest value in the grid problem and the second highest value in the others before exhausting the given resources. Nevertheless, the goal-directed approach provides very high value results often with much less time than the other approaches. It requires the least time in all domains except for the rover problem, but in this case, the large increase in value is likely worth the small increase in time. The other approaches cannot find a meaningful solution to the rover problem, while the goal-directed method can sometimes find the known optimal solution in the deterministic version (31.3809). Thus, the goal-directed approach is able to produce concise, high quality solutions that are based on the goal structure of the problem rather to producing a solution for an arbitrary fixed size.

6. CONCLUSIONS

In this paper, we discussed a natural class of problems in which agents have a set of joint goals. Under assumptions that each agent has a set of terminal actions and rewards for non-terminal actions are negative, we presented an indefinite-horizon model. To solve this model, we described a dynamic programming algorithm that is guaranteed to produce an optimal solution. For problems with

Two Agent Tiger Problem $|S| = 2$, $|A| = 3$, $|\Omega| = 2$

BFS	DEC-BPI	NLP	Goal-directed
-14.115 (3 nodes, 12007s)	-52.633 (11 nodes, 102s)	-1.088 (19 nodes, 6173s)[1]	5.041 (11,12 nodes, 75s)

Meeting in a Grid Problem $|S| = 16$, $|A| = 5$, $|\Omega| = 2$

BFS	DEC-BPI	NLP	Goal-directed
4.211 (2 nodes, 17s)	3.604 (7 nodes, 2227s)	5.658 (5 nodes, 117s)	5.637 (4 nodes, 4s)

Box Pushing Problem $|S| = 100$, $|A| = 4$, $|\Omega| = 5$

BFS	DEC-BPI	NLP	Goal-directed
-2 (1 node, 1696s)	9.442 (3 nodes, 4094s)	54.230 (4 nodes, 1824s)[1]	149.854 (5 nodes, 199s)

Rover Problems: deterministic and stochastic $|S| = 256$, $|A| = 6$, $|\Omega| = 8$

BFS	DEC-BPI	NLP	Goal-directed
x	-1.112 (3 nodes, 11262s)	9.642 (2 nodes, 379s)	26.932 (5 nodes, 491s)
x	-1.177 (3 nodes, 14069s)	8.119 (2 nodes, 438s)	21.483 (6 nodes, 956s)

Table 1: The values produced by each method along with controller size and time in seconds.

a more general reward structure or when other goal conditions are utilized rather than terminal actions, we modeled these problems as goal-directed DEC-POMDPs. We provided an approximate infinite-horizon approach for this class of problems and developed a bound on the number of samples needed to approach optimality. Unlike previous algorithms, our method uses the goal structure to produce finite-state controllers rather than optimizing controllers of an arbitrarily chosen size. Experimental results show that our approach can significantly outperform current state-of-the-art algorithms by quickly producing concise, high quality results even for large DEC-POMDPs.

In the future, we expect to extend our approach to non goal-directed problems by having a researcher define subgoals that may be useful for agents to achieve. Solving these smaller problems and combining the resulting controllers could allow much larger problems to be solved. We are also interested in exploring different ways of utilizing sampling to produce high quality results in other classes of DEC-POMDPs. With some modification, our sampling approach can be applied to general finite-horizon and infinite-horizon DEC-POMDPs and we would expect similarly high quality results in those cases.

7. ACKNOWLEDGEMENTS

Special thanks to Marek Petrik for his assistance with the analysis. Support for this work was provided in part by the National Science Foundation under grants IIS-0535061 and IIS-0812149, and by the Air Force Office of Scientic Research under grants FA9550-05-1-0254 and FA9550-08-1-0181.

8. REFERENCES

[1] C. Amato, D. S. Bernstein, and S. Zilberstein. Optimizing memory-bounded controllers for decentralized POMDPs. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, 2007.

[2] R. Becker, S. Zilberstein, V. Lesser, and C. V. Goldman. Solving transition-independent decentralized Markov decision processes. *Journal of AI Research*, 22:423–455, 2004.

[3] D. S. Bernstein, E. Hansen, and S. Zilberstein. Bounded policy iteration for decentralized POMDPs. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 1287–1292, 2005.

[4] D. S. Bernstein, S. Zilberstein, and N. Immerman. The complexity of decentralized control of Markov decision processes. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 32–37, 2000.

[5] C. V. Goldman and S. Zilberstein. Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of Artificial Intelligence Research*, 22:143–174, 2004.

[6] E. A. Hansen. Indefinite-horizon POMDPs with action-based termination. In *Proceedings of the Twenty-Second National Conference on Artificial Intelligence*, pages 291–296, 2007.

[7] E. A. Hansen, D. S. Bernstein, and S. Zilberstein. Dynamic programming for partially observable stochastic games. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, pages 709–715, 2004.

[8] M. Kearns, Y. Mansour, and A. Y. Ng. Approximate planning in large POMDPs via reusable trajectories. `http://robotics.stanford.edu/~ang/papers/pomdp-long.pdf`,1999.

[9] R. Nair, D. Pynadath, M. Yokoo, M. Tambe, and S. 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, 2003.

[10] S. D. Patek. On partially observed stochastic shortest path problems. In *Proceedings of the Fortieth IEEE Conference on Decision and Control*, pages 5050–5055, 2001.

[11] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: an anytime algorithm for POMDPs. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 1025–1032, 2003.

[12] S. Seuken and S. Zilberstein. Improved memory-bounded dynamic programming for decentralized POMDPs. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, 2007.

[13] T. Smith and R. Simmons. Heuristic search value iteration for POMDPs. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence*, 2004.

[14] D. Szer and F. Charpillet. An optimal best-first search algorithm for solving infinite horizon DEC-POMDPs. In *Proceedings of the Sixteenth European Conference on Machine Learning*, 2005.

[15] D. Szer, F. Charpillet, and S. Zilberstein. MAA*: A heuristic search algorithm for solving decentralized POMDPs. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, 2005.

[1]These results utilize controllers with deterministic action selection.