Heuristic Policy Iteration for Infinite-Horizon Decentralized POMDPs

Christopher Amato and Shlomo Zilberstein
Department of Computer Science
University of Massachusetts
Amherst, MA 01003 USA

Abstract

Decentralized POMDPs (DEC-POMDPs) offer a rich model for planning under uncertainty in multiagent settings. Improving the scalability of solution techniques is an important challenge. While an optimal algorithm has been developed for infinite-horizon DEC-POMDPs, it often requires an intractable amount of time and memory. To address this problem, we present a heuristic version of this algorithm. Our approach is able to use initial state information to decrease solution size and often increases solution quality over what is achievable by the optimal algorithm before resources are exhausted. Experimental results demonstrate that this heuristic approach is effective, producing higher values and more concise solutions in all three test domains.

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 affect jointly 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.

Recently, several approximate algorithms for solving DEC-POMDPs have been proposed [1, 4, 7, 9, 10, 13, 14, 16]. Of these, only [1, 4, 5, 14] address the infinite-horizon problem. One exception is an algorithm that can solve a subclass of infinite-horizon DEC-POMDPs, using an average reward formulation rather than the standard discounted objective. It is restricted however to *transition independent* problems with *full local observability* [10].

The infinite-horizon DEC-POMDP, in which the agents operate over some unbounded period of time, is particularly suitable for such domains as networking, management of agent communications, and some robot control problems, where the agents operate continuously.

The three approximate algorithms for solving infinite-horizon DEC-POMDPs use a common solution representation based on finite-state controllers to represent each agent's policy. The state of the controller determines the next action and the observation determines the transition to the next step. Some solution techniques restrict action selection, state transitions, or both to be deterministic in order to simplify the search space. Generally, stochastic controllers can yield higher values compared with deterministic ones of the same size, but they are also harder to derive.

The three existing algorithms seek concise controllers that optimize the objective function. These approaches perform well in some problems, but also have limitations. For instance, Amato et al.'s and Szer and Charpillet's approaches do not scale well to large controller sizes, which may be necessary for some problems. Bernstein et al.'s method [4] is more scalable, but it often produces low-quality results.

An ϵ-optimal algorithm for infinite-horizon DEC-POMDPs has also been developed [3]. This technique is an extension of policy iteration for POMDPs [7] in that it searches through policy space by a series of "backups" which improve the value of finite-state controllers. Rather than the single controller for the POMDP problem, a controller is used for each agent. At each step, the controllers are backed up and then nodes that are not useful for any problem state or any initial nodes of the other agents' controllers can be removed. These backup and pruning steps continue until the ϵ-optimal solution is returned. While this algorithm is provably optimal with a finite number of steps, it does not usually perform well in practice. The main reason for this is the amount of resources that are required by the approach. As backup cause controllers to often grows exponentially at each step an intractable amount of memory is soon required. Likewise, as the controller size grows, the linear programming that is used for the pruning step causes running time to quickly become unmanageable. As a result only a few backups can be completed and a high value is not yet achieved.

Because the optimal algorithm will not usually be able to return an optimal solution in practice, we introduce a heuristic version of policy iteration for DEC-POMDPs. This approach makes use of initial state information to prune many more nodes of the controller at each step and often results in better solution quality. This begins by sampling a desired number of belief states that are reachable from the initial state. On each pruning step, the highest-valued controller nodes for each of the sampled states are retained, thus allowing pruning to be conducted for all agents at the same time. This is an advantage for two reasons. First, only nodes that are useful for those sampled states that are likely to affect value of the controller are chosen. Second, by pruning all agents controllers at the same time, those nodes that are only useful for certain initial nodes of the other agents can also be removed. Conversely, when pruning is conducted for each agent separately, many nodes would be left even if better value could always be achieved without them. This process of backing up and pruning continues until value does not change. In practice, this often results in a more concise and higher-valued solution that what is produced by the optimal algorithm before resources are exhausted.

The rest of the paper is organized as follows. We first describe the DEC-POMDP model, its solution and the relevant previous work including the optimal policy iteration algorithm. We then present the heuristic policy iteration algorithm. Lastly, we provide experimental results of the optimal policy iteration and an improved implementation along with two variants of our heuristic algorithm on three problem domains. These results show that the optimal approach performs well, but the heuristic approach is able to construct more concise controllers with higher value on all three domains. This shows that our heuristic policy iteration can be used in practice to provide high quality policies to a wide range of DEC-POMDPs.

2 Background

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

2.1 The DEC-POMDP model

A two agent DEC-POMDP can be defined with the tuple: $M = \langle S, A_1, A_2, P, R, \Omega_1, \Omega_2, O \rangle$

- S, a finite set of states
- A_1 and A_2, finite sets of actions for each agent
- P, a set of state transition probabilities: $P(s'|s, a_1, a_2)$, the probability of transitioning from state s to s' when actions a_1 and a_2 are taken by agents 1 and 2 respectively
- R, a reward function: $R(s, a_1, a_2)$, the immediate reward for agent 1 taking action a_1 and agent 2 taking action a_2 while in system state s
- Ω_1 and Ω_2, finite sets of observations for each agent
- O, a set of observation probabilities: $O(o_1, o_2|s', a_1, a_2)$, the probability of agents 1 and 2 seeing observations o_1 and o_2 respectively given agent 1 has taken action a_1 and agent 2 has taken action a_2 and this results in state s'

Because we are considering the infinite-horizon problem, the decision making process unfolds over an infinite sequence of steps. 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 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 from each agent's perspective can also be considered by 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 agent. 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 solution. Each agent's policy can be represented as a local controller and the resulting set of controllers supply the joint policy, called the *joint controller*. 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, $\psi : Q \rightarrow \Delta A$ is the action selection model for each node, and $\eta : Q \times A \times O \rightarrow \Delta Q$ represents the node transition model for each node given an action was taken and an observation seen. For n agents, the value for starting in nodes $\vec{q}$ and state s is given by

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

This is also referred to as the Bellman equation. 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.

2.2 Approximate algorithms

Approximate algorithms that can solve infinite horizon DEC-POMDPs have been developed by Bernstein et. al [4], Amato et al. [1] and Szer and Charpillet [14]. Bernstein et al. and Amato et al. use linear programming and nonlinear programming (NLP) respectively to find parameters for stochastic finite-state controllers. Szer and Charpillet use best-first search to construct deterministic controllers.

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 algorithm 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.

In an attempt to address some of these problems, Amato et al. define a set of optimal controllers given a fixed size with a nonlinear program. Because it is often intractable to solve this NLP optimally, a locally optimal solver is used. Unlike DEC-BPI, this approach allows start state information to be used so smaller controllers may be generated and improvement takes place in one step. While, concise controllers with high value can be produced, large controllers, which may be required for some problems, cannot produced by the current locally optimal solvers.

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. They prove that this technique will generate the optimal deterministic finite state controller of a given size, but its use remains limited due to high resource requirements. As a consequence, only very small controllers can be produced and thus the approach lacks the scalability to provide high quality solutions.

Algorithm 1: Optimal Policy Iteration

```
input  : A set of arbitrary one node controllers for each agent, $Q_0$
output: An $\epsilon$-optimal set of controllers $Q^*$
begin
    $t = 0$
    while $\frac{\gamma^{t+1}|R_{max}|}{1-\gamma} > \epsilon$ do
        evaluate($Q_t$)
        $Q_t \leftarrow$ exhaustiveBackup($Q_t$)
        $Q_t \leftarrow$ transform($Q_t$)
        $t \leftarrow t + 1$
    return $Q_t$
end
```

2.3 Optimal Policy Iteration

The optimal DEC-POMDP algorithm [3] is a dynamic programming approach which generalizes policy iteration for infinite-horizon POMDPs [6]. Like the POMDP algorithm, a controller for each agent is constructed over a series of steps. At each step, an exhaustive backup is performed where new nodes are added to each agent's controller for all possible action and observation pairs. That is, a controller with $|Q_i|$ nodes before this backup has $|A_i||Q_i|^{|\Omega_i|}$ nodes afterward. If these backups continue to take place, the controllers will be able to produce a value within ϵ of the optimal value for the given problem. This is because the procedure would result in a brute force search through deterministic policy space. While this will eventually converge to ϵ-optimality, it is also extremely inefficient.

In order to improve performance, *value-preserving transformations* are used. These transformations allow nodes to be removed or the parameters of the controllers to change as long as there is is a mapping that results in no loss of value. More formally, a transformation is defined as value-preserving for joint controllers C and D if $C \leq D$ where $\leq$ is defined as follows

Definition 1. **Value-Preserving Transformation:** Consider joint controllers C and D with node sets Q and R respectively. We say that $C \leq D$ if there exists a mapping $f_i : Q_i \rightarrow \Delta R_i$ for each agent i such that $V(s, \vec{q}) \leq \sum_{\vec{r}} P(\vec{r}|\vec{q})V(s, \vec{r})\ \forall s \in S, q \in Q$

One such transformation permits nodes of the controllers to be removed, or pruned, after each exhaustive backup without losing the ability to eventually converge to an ϵ-optimal policy. The linear program and its dual that are used to accomplish this pruning are shown in Table 1. The linear program tests to see if there is some distribution over the policies of the other agents and states of the system for which starting in node q in agent i's controller has equal or higher value than starting in any of the other nodes in the controller. If no such generalized belief state can be found, the node is considered dominated and may be removed from the controller. The dual then describes how to adjust the dominated node's incoming links. The dual's solution provides a distribution over nodes of the agent's controller that have equal or higher value than the dominated node starting in any state and given any

Primal – For variables: ϵ, $x(q_{-i}, s)$, Maximize ϵ

Given the improvement constraints: $\forall \hat{q}_i \sum_{q_{-i},s} x(q_{-i}, s)\Big[V(\vec{q}, s) - V(\hat{q}_i, q_{-i}, s)\Big] \geq \epsilon$

And probability constraints: $\sum_{q_{-i},s} x(q_{-i}, s) = 1$ and $\forall s, q_{-i}\ x(q_{-i}, s) \geq 0$

Dual – For variables: ϵ, $x(\hat{q}_i)$, Minimize ϵ

Given the improvement constraints: $\forall s, q_{-i} \sum_{\hat{q}_i} x(\hat{q}_i)V(\hat{q}_i, q_{-i}, s) + \epsilon \geq V(\vec{q}, s)$

And probability constraints: $\sum_{\hat{q}_i} x(\hat{q}_i) = 1$ and $\forall \hat{q}_i\ x(\hat{q}_i) \geq 0$

Table 1: The linear program used to determine if there is a generalized belief point at which node q for agent i is undominated. In the primal, variable $x(q_{-i}, s)$ represents $P(q_{-i}, s)$, the probability that the other agents use the set of policies q_{-i} and the system state is s. In the dual, $x(\hat{q}_i)$ represents $P(\hat{q}_i)$, the probability of starting in node $\hat{q}$ for agent i.

initial nodes for the other agents' controllers. Since this distribution has higher value for any generalized belief state, it is always better than the dominated node and may safely replace it.

The policy iteration algorithm is shown in Algorithm 1. Alternating between exhaustive backups and pruning will eventually allow an ϵ-optimal controller to be generated. Convergence can be determined by the number of backups performed and the discount factor of the problem. That is, the algorithm may terminate after step t if $\frac{\gamma^{t+1}|R_{max}|}{1-\gamma} < \epsilon$ where $|R_{max}|$ is the maximum reward in the problem. This test ensures that any additional rewards after t steps will be sufficiently small to still guarantee ϵ-optimality. More detail and a proof of convergence can be found in [3].

As mentioned above, any value-preserving transformation or combination of transformations may be used after an exhaustive backup. While pruning was described, another possibility is to use DEC-BPI as well to more quickly improve the value of a controller. On each step of DEC-BPI a single node is improved by finding a probability distribution over actions for that node and transitions to other nodes of the controller. Because changes are only made if value improves over all states of the problem and initial nodes of the other agents' controllers this is a value-preserving transformation. The incorporation of DEC-BPI in policy iteration may allow higher value controllers to be generated in fewer steps and an increased number of nodes may also be able to be pruned.

3 Heuristic Policy Iteration

The central features of our approach are that it is able to prune nodes for all agents simultaneously and uses a set of belief points based on a centralized search starting from the initial system state. Therefore, we first discuss the benefits of simultaneous pruning and how centralized belief points could be sampled. After this we provide the motivation and details of our heuristic algorithm.

3.1 Simultaneous pruning

Pruning, as defined for the optimal algorithm, will not remove all nodes that could be removed from all the agents' controllers. Because pruning requires each agent to consider the controllers of other agents, after nodes are removed for one agent, the other agents may be able to prune other nodes. Thus pruning must cycle through the agents and ceases when no agent can remove any further nodes. This is both time consuming and causes the controller to be much larger than it needs to be.

As an alternative, if nodes could be removed for all agents at the same time, many more nodes could be removed. We refer to any method that prunes all the agents' controllers at the same time as *simultaneous pruning*. An example of the usefulness of simultaneous pruning is shown in Figure 1. The extremely simple two agent DEC-POMDP in this example has one state, two actions and one observation. The rewards are given by: $R(s, a_1, a_1) = 2$, $R(s, a_2, a_1) = -10$, $R(s, a_1, a_2) = 1$, $R(s, a_2, a_2) = 3$. The transition and observation probabilities are trivially 1 for the one state and one observation.

The figure shows a one node initial controller after an exhaustive backup. The initial (and optimal) controller consists of each agent taking action 2 each step and because the optimal algorithm would not yet know that it had found the optimal controller a backup was performed. Using the pruning of the optimal policy iteration algorithm, as defined in Table 1, will remove the duplicate node of taking action 2 and then transitioning into the initial controller (shown in light gray in the figure), but will retain the other node of taking action 1 first. This can be seen by looking at the values of each node in an agent's controller given each of the nodes of the other agent's controller. That is, if we term the node that takes action 1 q_1 and the node that takes action 2 q_2, for discount γ, $V(s, q_1, q_1) = 2 + 3\gamma/(1-\gamma)$, $V(s, q_1, q_2) = 1 + 3\gamma/(1-\gamma)$, $V(s, q_2, q_1) = -10 + 3\gamma/(1-\gamma)$ and $V(s, q_2, q_2) = 3/(1-\gamma)$. As pruning is conducted one agent at a time, q_1 appears useful for each agent. This is because if the other agent begins executing its controller in q_1, the agent is always better off by beginning in q_1 itself.

By using simultaneous pruning, the extraneous q_1 node could also be removed. We already know that q_1 is not part of the optimal controller, so it cannot be useful to retain. To see this we can define a value-preserving transformation that maps node q_2 to itself and node q_1 to node q_2 in each agents' controller. The value of the resulting transformation is $3/(1-\gamma)$ which is at least as high as any of the values produced by any of nodes in the previous controller. We see that unless we remove node q_1 from each agent's controller at the same time, the extraneous node will remain. This will be compounded as more backups are performed leading to controllers that have many more nodes than are necessary to represent an ϵ-optimal value. In general, simultaneous pruning can allow more steps of backing up and pruning to occur and could lead to producing optimal or near-optimal controllers with much less time and memory.

3.2 Centralized planning

If planning and execution are centralized, a DEC-POMDP can be transformed into a POMDP [12]. In this case, the observations of all agents are known to a central decision maker and the best actions can be taken based on full knowledge of the observations

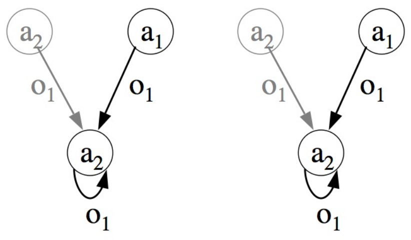

Figure 1: Example of controllers with extraneous nodes that can only be removed with simultaneous pruning

Algorithm 2: Heuristic Policy Iteration

```
input  : A set of arbitrary one node controllers for each agent, $Q_0$ and the number of
         desired belief points, $k$
output: An improved set of controllers $Q^h$
begin
    $B \leftarrow$sampleBeliefPoints($b_0$)
    $value_0 \leftarrow$ evaluate($Q_0$)
    while $value_t - value_{t-1} > \epsilon$ do
        $Q_t \leftarrow$ exhaustiveBackup($Q_t$)
        $Q_t \leftarrow$ prune($Q_t, B$)
        $value_t \leftarrow$ evaluate($Q_t$)
        $t \leftarrow t + 1$
    return $Q_t$
end
```

that are seen. Similarly, if each agent communicates its observation on each step to all other agents, belief states can be generated and the problem can be solved as a POMDP. That is, starting with the initial state, if an action is taken and an observation seen, the next belief state can be calculated. So from any belief state, given an action taken and an observation seen, the resulting belief state is known.

If a search of the state space is conducted for a POMDP based on some policy and starting at the initial state a set of belief points can then be generated. This is the basis for point-based solvers, such as [8, 11]. The performance of these methods depends on the number and distribution of points that are chosen. If a large enough set of points is used or the points are those that would be visited by an optimal policy, an ϵ-optimal solution can be found.

3.3 Heuristic pruning

While the optimal policy iteration method guarantees that a controller arbitrarily close to optimal will be found, the controller may be very large and many unnecessary nodes may be generated along the way. This is exacerbated by the fact that the algorithm cannot take advantage of an initial state distribution and must attempt to improve the controller for any initial state. As a way to combat these disadvantages, we have developed a heuristic

For variables: ϵ, $x(\hat{q}_i)$ and for each belief point b, Maximize ϵ

Given the improvement constraints: $\forall b, q_{-i} \;\; \sum_s b(s)\Big[\sum_{\hat{q}_i} x(\hat{q}_i)V(\hat{q}_i, q_{-i}, s) - V(\vec{q}, s)\Big] \geq \epsilon$

And probability constraints: $\sum_{\hat{q}_i} x(\hat{q}_i) = 1$ and $\forall \hat{q}_i \; x(q_i) \geq 0$

Table 2: The linear program used to determine if a node q for agent i is dominated at each point b and all initial nodes of the other agents' controllers. As node q may be dominated by a distribution of nodes, variable $x(\hat{q}_i)$ represents $P(\hat{q}_i)$, the probability of starting in node $\hat{q}$ for agent i.

version of policy iteration that prunes nodes based on their value only at a given set of belief points. As a result, the algorithm will no longer be optimal, but it can often produce more concise controllers with higher solution quality for a given start state.

To produce the set of belief points, we assume each agent shares its observations as described in the previous subsection. Thus, for a given policy for the agents, a sampled set of centralized belief states can be found. Recall that planning can be completed centrally in order to generate an independent controller for each agent which is used to act distributively. Therefore, this assumption can be used in the planning phase as long as independent controllers are generated for execution. When decision making is decentralized, each agent can no longer calculate a belief state for each step of the problem (beyond the first). If the observations for the other agents could be seen, a centralized belief state could be calculated and if the search was done well, the centralized belief state would be one of those that was found. So, while a belief state cannot actually be calculated by an agent that is acting distributively, the set of points should still give useful information about what observations the other agents could have seen.

Once the set of belief points has been found, it is used to guide the pruning process of our algorithm. There are two main advantages of this approach: it allows simultaneous pruning and it focuses the controller on certain areas of the belief space. The benefits of simultaneous pruning were discussed above and it is worth noting that unlike the optimal approach, our heuristic algorithm would not keep the extraneous nodes in the example shown in Figure 1 and it would converge to the optimal controller after one backup.

The advantage of considering a smaller part of the state space has already been shown to produce drastic performance increases in POMDPs [8, 11] and finite-horizon DEC-POMDPs [13, 15]. For POMDPs, a problem with many states has a large dimensional state space, but many parts may never be visited by an optimal policy. Focusing on a subset of belief states can allow a large part of the state space to be ignored without significant loss of solution quality. The problem of having a large state space is compounded in the DEC-POMDP case. Not only is there uncertainty about the state, but also about the policies of the other agents. As a consequence, the generalized belief space which includes all possible distributions over states of the system and current policies of the other agents must often be considered to guarantee optimality. This results in a huge space which contains many unlikely states and policies. Belief updates cannot be calculated for DEC-POMDPs, but considering likely states allows likely policies for all agents to be chosen and if done correctly may provide enough information to allow an optimal or near-optimal solution to be found.

We provide a formal description of our approach in Algorithm 2. The desired number of belief points, k, is chosen and that number is found by centralized search while assuming random policies for each agent. The search begins at the initial state of the problem and continues until the given number of points is obtained (with the possibility of being reset if necessary). The arbitrary initial controller is also evaluated and the value at each state and for each initial node of any agent's controller is retained. While the (maximum) value at any state changes more than some parameter ϵ, the controller is backed up and pruned. The exhaustive backup procedure is exactly the same as the one used in the optimal algorithm. Pruning the controller takes place in two steps. First, for each of the k belief points, the highest-valued set of initial nodes is found. To accomplish this, the value of each combination of nodes for all agents is calculated for each of these k points and the best combination is kept. This allows nodes that do not contribute to any of these values to be quickly removed. Next, each node of each agent is pruned using the linear program shown in Table 2. If a distribution of nodes for the given agent has higher value at each of the belief points for any initial nodes of the other agents' controllers, it is pruned and replaced with that distribution. The new controllers are then evaluated and the value is compared with the value of the previous controller. These steps continue until convergence is achieved.

Similar to how DEC-BPI can be used to improve solution quality for optimal policy iteration, the nonlinear programming approach of Amato et al. can be used to improve solution quality for the heuristic case. To accomplish this, instead of optimizing the controller for just the initial belief state of the problem, all the belief points being considered are used. A simple way to do this is to maximize over the sum of the values of the initial nodes of the controllers weighted by the probabilities given for each point. This approach can be used after each pruning step and may provide further improvement of the controllers.

4 Experiments

To evaluate the performance of optimal and heuristic policy iteration, we tested two variants of each on three different domains. These variants consist of solely pruning after each backup and incorporating additional improvement. For the optimal policy iteration algorithm improvement consists of using DEC-BPI after pruning has been completed and for the heuristic version, the nonlinear program (NLP) for the set of sampled belief points is optimized. These four implementations were all initialized with one node controllers consisting of the first action in the problem definition (which transitions to itself). The values and running times are reported after each algorithm completes a given number of steps, which are considered to be a single iteration of the while loop in Algorithms 1 and 2. All algorithms were run on a Intel Core 2 Duo 3.4GHz with 2GB memory, but because of a lack of a NLP solver, the NLP technique was run on the NEOS server. Only the running time returned by the server was added in this case and not the additional overhead of writing and uploading the file, waiting in the queue and parsing the result.

As mentioned above, three domains were used to test the algorithms. The first domain is called the multiagent tiger problem and consists of 2 states, 3 actions and 2 observations

Two Agent Tiger Problem

step	Policy Iteration (PI)	PI with DEC-BPI	Heuristic PI (HPI)	HPI with NLP
0	-150 (1,1 in 1s)	-150 (1,1 in 1s)	-150 (1,1 in 1s)	-20 (1,1 in 1s)
1	-137 (3,3 in 1s)	-20 (3,3 in 12s)	-137 (2,2 in 1s)	-20 (3,3 in 24s)
2	-117.9 (15,15 in 7s)	-20 (15,15 in 201s)	-117.9 (4,6 in 2s)	-9.3 (5,5 in 602s)
3	-98.9 (255,255 in 1433s)	-20 (255,255 in 3145s)	-103.5 (6,8 in 7s)	-8.7* (7,7 in 2257s)
4	x	x	-95.2 (7,9 in 26s)	x
...	...	...	...	...
15	x	x	-43.3 (18,21 in 4109s)	x

Meeting in a Grid Problem

step	Policy Iteration (PI)	PI with DEC-BPI	Heuristic PI (HPI)	HPI with NLP
0	2.8 (1,1 in 1s)	2.8 (1,1 in 1s)	2.8 (1,1 in 1s)	3.4 (1,1 in 2s)
1	3.4 (5,5 in 7s)	3.8 (5,5 in 452s)	3.74 (4,3 in 112s)	4.3 (3,5 in 946s)
2	3.7 (80,80 in 2402s)	3.8* (80,80 in 37652s)	3.9 (5,10 in 369s)	6.2* (7,9 in 6157s)
3	x	x	4.0 (11,11 in 3073s)	x
4	x	x	4.0 (14,16 in 6397s)	x
5	x	x	4.1 (23,24 in 54296s)	x

Box Pushing Problem

step	Policy Iteration (PI)	PI with DEC-BPI	Heuristic PI (HPI)	HPI with NLP
0	-2 (1,1 in 4s)	-2 (1,1 in 53s)	-2 (1,1 in 4s)	-1.7 (1,1 in 44s)
1	-2 (2,2 in 254s)	9.4 (2,2 in 6909s)	-2 (2,2 in 182s)	32.8 (2,2 in 2018s)
2	12.8 (9,9 in 31225s)	x	28.7 (3,2 in 1860s)	53.1 (3,2 in 25476s)
3	x	x	35.6 (4,4 in 24053s)	x

Table 3: Values for the initial state of each problem are given for policy iteration with and without DEC-BPI and our heuristic policy iteration with and without optimizing the nonlinear program (NLP) defined by Amato et al. These values are given for a range of backup and pruning steps along with the size of each agent's controller and the running time in parentheses. Five belief points were used for the tiger problem, ten for the grid problem and 20 for the box problem. An asterisk (*) is used to donate that backup, pruning and evaluation were completed, but the added improvement of DEC-BPI or the NLP could not be completed.

[9]. In this problem, there are two doors. Behind one door is a tiger and behind the other is a large treasure. Each agent may open one of the doors or listen. If either agent opens the door with the tiger behind it, a large penalty is given. If the door with the treasure behind it is opened and the tiger door is not, a reward is given. If both agents choose the same action (i.e., both opening 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 seen 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 one of the door with equal probability.

The second domain is somewhat larger. It is a grid problem that consists of 16 states, 5 actions and 2 observations [4]. In this problem, two agents must meet in a 2x2 grid with no obstacles. The agents start 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 and a reward of 1 is given for each step the agents share the same square.

The third domain is much larger. This problem, with 100 states, 4 actions and 5 observations consists of two agents that can gain reward by pushing different boxes [13]. The agents begin facing each other in the bottom corners of a 4x3 grid with the available actions of turning right, turning left, moving forward or staying in place. There is a 0.9 probability that the agent will succeed in moving and otherwise will stay in place, but the two agents can never occupy the same square. The middle row of the grid contains two small boxes and one large box. This large box can only be moved by both agents pushing at the same time. The upper row of the grid is considered the goal row. The possible deterministic observations consist of seeing an empty space, a wall, the other agent, a small box or the large box. A reward of 100 is given if both agents push the large box to the goal row and 10 is given for each small box that is moved to the goal row. A penalty of -5 is given for each agent that cannot move and -0.1 is given for each time step. Once a box is moved to the goal row, the environment resets to the original start state.

The results from these domains are displayed in Table 3. The value for the given initial state is shown along with the number of nodes in each agent's controller and the running time required to produce the solution. For the two agent tiger problem five belief points were used, ten were used for the meeting in a grid problem and twenty were used in the box pushing problem. These quantities were chosen experimentally for small step numbers, but an exhaustive search was not performed and other values could produce better results. These results are given for the number steps that the algorithm is able to complete.

It can be seen that for each number of steps our heuristic policy iteration algorithm which is improved with Amato et al.'s NLP has higher value than all the other algorithms. It also produces the highest value that is achievable for each problem. Heuristic policy iteration without improvement with the NLP also performs very well. It is able to outperform policy iteration for all but one number of steps and produces the second highest value, regardless of number of steps, on two problems. Also, while using the NLP approach can be time consuming, the regular heuristic algorithm is very fast. Because a small set of points is considered there is much less computation that is required, causing it to be the fastest algorithm that was tested. Policy Iteration (PI) often had the second fastest running times usually followed PI with DEC-BPI and Heuristic PI with the NLP. The running times of these last two algorithms did not show that one was clearly faster than the other.

The heuristic algorithms also provide more concise controllers than their counterparts. This allows heuristic policy iteration to conduct many more backup and pruning steps and still provide controllers that are a fraction of the size of those produced by Bernstein et al.'s policy iteration techniques. The optimal algorithms quickly generate very large controllers which causes them to exhaust the available resources and are no longer able to conduct backups or pruning. Thus, the heuristic approaches show that they are able to provide much smaller solutions that have higher value than Bernstein et al.'s approaches.

5 Conclusions

In this paper, we introduced a more scalable dynamic programming algorithm for solving infinite-horizon DEC-POMDPs, called heuristic policy iteration. Our algorithm allows initial state information to be used to focus solutions on a sampled set of points. We showed that, in practice, our heuristic version outperforms Bernstein et al.'s optimal policy iteration algorithm. This occurs because while Bernstein et al.'s algorithm is optimal in theory, it quickly exhausts the available resources before it can produce an optimal solution. We tested two versions of our heuristic approach against two versions of Bernstein et al.'s approach in three diverse domains. In all three of these domains, significantly higher values were found with much smaller controllers by one of our heuristic approaches. Our results show that our heuristic algorithms can produce more concise, higher-valued controllers often in less time than Bernstein et al.'s approaches.

In the future, we plan to explore different methods to sample points and ways to grow the controller without performing exhaustive backups. While we currently use random policies for each agent, an increase in performance may be possible by choosing policies in a more sophisticated manner. One method would be to use a heuristic policy that has better solution quality than a random policy, but different sampling techniques may perform better. Exhaustive backups are one of the main causes of exhaustion of resources in all of the algorithms that we tested. Nodes must be added for all possible action and observation pairs, representing an exponential increase in each agent's controller size. Only after this exponential backup can pruning take place. In POMDPs, a method has been developed to interleave backing up and pruning [2], but a similar method has not yet been developed for DEC-POMDPs. Development of this type of method may allow the optimal and heuristic policy iteration algorithms to produce much larger controllers and thus higher quality solutions.

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 Uncer-*

tainty in Artificial Intelligence, Vancouver, Canada, 2007.

[2] N. L. Z. Anthony Cassandra, Michael L. Littman. Incremental pruning: A simple, fast, exact method for partially observable markov decision processes. In *uai97*, San Francisco, CA, 1997.

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

[4] 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, Edinburgh, Scotland, 2005.

[5] R. Emery-Montemerlo, G. Gordon, J. Schneider, and S. 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.

[6] E. A. Hansen. Solving POMDPs by searching in policy space. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 211–219, Madison, WI, 1998.

[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, San Jose, CA, 2004.

[8] S. Ji, R. Parr, H. Li, X. Liao, and L. Carin. Point-based policy iteration. In *Proceedings of the Twenty-Second National Conference on Artificial Intelligence*, Vancouver, Canada, 2007.

[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, Acapulco, Mexico, 2003.

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

[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*, Acapulco, Mexico, 2003.

[12] S. Seuken and S. Zilberstein. Formal models and algorithms for decentralized control of multiple agents. Technical Report UM-CS-2005-068, University of Massachusetts, Department of Computer Science, Amherst , MA, 2005.

[13] 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*, Vancouver, Canada, 2007.

[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*, Porto, Portugal, 2005.

[15] D. Szer and F. Charpillet. Point-based dynamic programming for DEC-POMDPs. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, Boston, MA, 2006.

[16] 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*, Edinburgh, Scotland, 2005.