Bounded Dynamic Programming for Decentralized POMDPs

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

ABSTRACT

Solving decentralized POMDPs (DEC-POMDPs) optimally is a very hard problem. As a result, several approximate algorithms have been developed, but these do not have satisfactory error bounds. In this paper, we first discuss optimal dynamic programming and some approximate finite horizon DEC-POMDP algorithms. We then present a bounded dynamic programming algorithm. Given a problem and an error bound, the algorithm will return a solution within that bound when it is able to solve the problem. We give a proof of this bound and provide some experimental results showing high quality solutions to large DEC-POMDPs for large horizons.

1. INTRODUCTION

Cooperative multiagent problems that include uncertainty are very common, but are often difficult to solve optimally. The decentralized partially observable Markov decision process (DEC-POMDP) is an extension of the POMDP framework to model these mutiagent settings. Each agent must make decisions based on imperfect information of the system state and uncertainty about the other agents. Solutions seek to maximize shared reward using solely local information for each agent.

Applications of the DEC-POMDP model include robot control [2] and networking [4]. Robots typically have sensors that provide uncertain and incomplete information about the state of the environment and the location of the other robots. This lack of information must be factored into the planning process. In a decentralized network, each node must make decisions about when and where to send packets without full knowledge of the structure or actions of the rest of the network. Other applications include e-commerce and space exploration systems.

Several exact and approximate algorithms have recently been developed to solve finite-horizon DEC-POMDPs [2, 4, 5, 6, 7]. The finite-horizon problem, in contrast to the infinite-horizon version, takes place over a finite number of steps. Current exact algorithms that use dynamic programming often require an intractable amount of space even though an optimal or near-optimal solution may be concise. DEC-POMDP approximation algorithms can operate with a limited amount of memory, but as a consequence may provide solutions that are unboundedly below the optimal. We propose a new approach that addresses the space requirement of DEC-POMDP algorithms while maintaining error bounds on the solution quality. This approach uses a bounded approximation of the value function for each agent that may require less space than the representation used by optimal DEC-POMDP algorithms. This technique has already been successful in POMDP domains [3, 8], and we extend this idea to the DEC-POMDP case. Bounded error dynamic programming permits near-optimal solutions to be found for much larger DEC-POMDPs and horizons.

In this paper, we first present some background on the DEC-POMDP model. We then discuss some previous approaches to solve DEC-POMDPs and our ϵ-pruning approach that is able to overcome some of the shortcomings of the earlier work. Lastly, experimental results are provided comparing our approach with a state-of-the-art approximate algorithm. We show ϵ-pruning allows high quality solutions to be found that are within a given bound of optimal for a range of domains and horizons.

2. THE DEC-POMDP MODEL

We first review the decentralized partially observable Markov decision process (DEC-POMDP) model. For clarity, we present the model for two agents as it is straightforward to extend it to n agents.

A two agent DEC-POMDP can be defined with the tuple:

$M = \langle S, A_1, A_2, P, R, \Omega_1, \Omega_2, O, T \rangle$

- S, a set of states
- A_1 and A_2, sets of actions for each agent
- P, state transition probabilities: $P(s'|s, a_1, a_2)$, the probability of transitioning to state s' given the previous state was s and actions a_1 and a_2 were taken by agents 1 and 2 respectively
- R, a reward function: $R(s, a_1, a_2)$, the immediate reward for being in state s and agent 1 taking action a_1 and agent 2 taking action a_2
- Ω_1 and Ω_2, sets of observations for each agent

MSDM 2007 May 15, 2007, Honolulu, Hawai'i, USA.

- O, an 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 causing the state to transition to s'
- T, a horizon of the problem

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. Because the true state is unknown, local observation histories may be needed to make an optimal action choice. 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 that is executed by that agent. The goal in the finite-horizon problem is to maximize the total cumulative reward over the given horizon, beginning at some distribution over states called a *belief state*. An *optimal joint policy* for a certain belief state is the joint policy with the maximum value at that belief state. We express this as $\max_{p,q} \sum_s b(s)V(p, q, s)$ for all possible policies p and q of horizon T, summing over all states s.

We can model DEC-POMDP policies as policy trees. In a policy tree, the root node defines what action is taken by that agent on the first step of the problem. The arcs from that node represent observations, which lead to new action nodes given that observation. This continues until a leaf is reached, producing the action for the last step of the problem. The depth of the tree corresponds to the horizon of the problem. Each agent's policy can be described as a policy tree independent of the other agents.

In the two-agent problem, the value of an agent's policy tree, p, can be determined based on a policy for the other agent, q, and the initial state, s. This can be written recursively as:

$$V(p, q, s) = R(s, a_p, a_q) +$$

$$\sum_{s'} P(s'|a_p, a_q, s) \sum_{o_1,o_2} O(o_1, o_2|s', a_p, a_q)V(p_{o_1}, q_{o_2}, s')$$

where a_p and a_q represent the root actions taken in the policy trees of each agent and $V(p_{o_1}, q_{o_2}, s')$ is the value of continuing in the policy trees based on the observation that was seen and the resulting state.

3. OPTIMAL DP

Hansen et al. [4] have developed an optimal dynamic programming (DP) algorithm for DEC-POMDPs. The authors noticed that if communication is not assumed, each agent must choose policies that have high value based on any policy the other agent may use. To formalize this, they define a generalized belief state, $b(s, q)$, for an agent as the probability that the system state is s and the other agent will use policy q. The value of an agent's belief is then:

$$V(b) = \max_p \sum_{s,q} V(p, q, s)b(s, q)$$

which represents the maximum the agent can achieve given its set of policies weighted by the probability that the true state is s and the other agent uses policy q. As the probability distribution of the underlying state depends on the policies of the other agent and we assume no prior knowledge about the other agent, all policies that may contribute to this max must be retained. This suggests that policies that do not contribute to the max may be removed, or *pruned*. Policy trees for horizon two and a visual representation of the generalized belief space is shown in Figure 1.

Hansen et al. generalize dynamic programming for the POMDP case and build up a policy tree for each agent one step at a time. They present a centralized algorithm that creates policy trees for each agent that can then be executed distributively. Sets of policy trees are grown for each agent from the last step of the problem to the first. The algorithm initializes these sets to include policies of horizon 1, which are merely the sets of actions for each agent. The actions that have lower value for all possible states and actions of the other agents are then removed. This is possible because by the time this step is reached in the final tree, trees that contribute to the max must have higher value than any other tree for some generalized belief state. Since there are multiple agents in the pruning process, each agent prunes its set of horizon 1 policies given the current sets for the other agents. As the other agents' sets of polices may be reduced in size as they are pruned, this may affect which policies can be pruned for other agents. Because of this, pruning continues until no agent is able to prune any of its remaining policies. Next, policy trees for all agents are grown to allow all possible actions to be taken and all possible observations to then be seen. Each action/observation pair then transitions to the policy trees from the previous step. This is referred to as a backup and the resulting increase in the number of trees for each agent is exponential. Next, trees which have lesser value for all possible states and policies of the other agents are pruned as described above. The growing and pruning process continues until the trees reach the given horizon. This results in a set of policy trees for each agent that is optimal for any initial initial state.

While this method guarantees that an optimal tree will be found, the set of trees may be very large and many unnecessary trees may be generated along the way. Because of this, it requires an intractable amount of memory for any but the smallest problems.

4. APPROXIMATE METHODS

As an alternative to exact methods, several DEC-POMDP approximation algorithms have been developed. Two of the best techniques are those of Nair et al.[5] and Emery-Montemerlo et al.[2] Both algorithms iterate through the set of agents, holding all policies fixed except one and choosing the best policy for that agent. This continues until there is no change in the policies selected. While this approach will allow a locally optimal solution to be found, it may be very poor and still requires the space of all policies for a single agent to be searched. Nair et al. propose a way to scale up their algorithm to larger problems by incorporating dynamic programming, but effects are limited in practice. Emery-Montemerlo et al. scale up their algorithm by approximating the problem by a series of simpler ones. While this increases solvable problem size, it decreases solution quality.

Seuken and Zilberstein [6] devised a heuristic approach to allow solutions to be found for arbitrary horizons. The authors use a forward search heuristic based on the start state of the problem to determine likely belief states at later states of the problem. This allows the number of trees kept

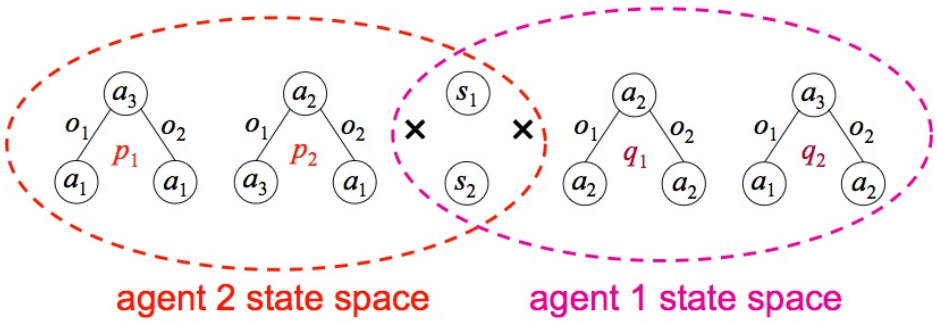

Figure 1: Policy trees for two agents and their resulting generalized state space that depends not only on the states of the system, but also the policy of the other agent

for each agent to be bounded at each step, but maintains high quality solutions. While Seuken and Zilberstein are able to demonstrate good time and value performance for some small DEC-POMDPs, their algorithm may return solutions that are unboundedly below the optimal solution. Also, all of the above authors assume knowledge of the start state in order to improve performance. As this knowledge is not always present or appropriate, we concentrate on finding the best solutions for any possible initial state.

5. BOUNDED DP

The main limitation of the optimal approach of Hansen et al. is that too many trees are retained at each step, resulting in huge memory requirements for all but the simplest problem and the shortest horizons. In order to mitigate this problem, we introduce a small error term, ϵ, and we prune policies that are higher valued than the remaining policies in the set by at most ϵ. This may allow a smaller set of policies to be used in the next step. The tradeoff is that there is a possible loss in value of at most ϵ from the optimal value at any belief state. We will refer to this method as *ϵ-pruning*.

5.1 ϵ-pruning approach

To understand how ϵ-pruning works, we go into more detail on how pruning is done by Hansen et al. Each policy can be represented as a vector of values with each component corresponding to the value of following the policy tree beginning at a given state. The value of each generalized belief state is given by the internal points of the vector. An example of this is shown in Figure 2 in the context of pruning and ϵ-pruning.

The authors iteratively construct a set of vectors that will not be pruned, called the undominated set. The set is initialized with the policy that has the highest value for an arbitrary belief state. The belief state that is used in pruning is the generalized belief state that is a probability distribution not only over the states of the problem, but also over the policies of the other agent(s). The remaining policies are examined one at a time and compared to the current undominated set using a linear program. We refer to a policy as dominated if the linear program shows that it does not improve on any undominated policy for any possible distribution of belief states. If the policy that is being examined is dominated, it is pruned. If it is undominated, the best tree for the belief at which it dominates is added to the undominated set. The linear program to determine dominance is shown in Figure 1. If δ is greater than 0, there is some belief state for which the policy has higher value than the undominated set. After the agent has tested each of its policies, the resulting undominated set is then used as part of the state space for the other agents. Pruning continues for each agent until no agent further reduces their undominated set.

Instead of merely pruning, however, we extend the ϵ-pruning technique to the DEC-POMDP case. This idea has been incorporated into POMDP algorithm implementations and described in [3, 8]. In general, in ϵ-pruning, a bound is added to the test of dominance for each policy tree. If the resulting δ in the linear program is less than our ϵ bound, it is considered to be dominated and we refer to such a policy as ϵ-dominated. An example of this can be seen in Figure 2. In this case, pruning retains four policy trees and their respective vectors for the given agent. The maximum loss is given by the δ value in the above linear program. This shows how dominant a given vector is with respect to the other vectors in the set. As the figure shows, if vector 2 is compared with the other vectors in the set, a small δ results. For this reason, if ϵ-pruning is used with $l_4 > \epsilon \geq l_2$ vector 2 is also pruned, as shown. For the POMDP case, the belief space considered is all distributions over states. In the next section, we show that ϵ-pruning can be used in the context of DEC-POMDPs by using the generalized belief state.

5.2 Bounding total error

More formally, we define one step of ϵ-pruning for the DEC-POMDP case as one pruning step by one agent. Policies are considered dominated if δ is less than ϵ in the check for dominance above. This definition is necessary due to the iterated pruning that is often required for the agents to converge to a fixed number of policies. ϵ-pruning can be performed for one agent any number of times and by any number of agents, but as we will see, each time it is performed, we may lose ϵ value from an optimal solution. The goal of ϵ-pruning is to reduce the amount of memory necessary to represent an optimal or near optimal value for any state distribution. That is, ϵ-pruning attempts to reduce the number of policies retained while producing a close approximation to the optimal joint policy for any initial distribution over states.

THEOREM 1. *One step of ϵ-pruning for one agent results in joint policy values that are at most ϵ below the optimal joint policy value for any initial distribution over states.*

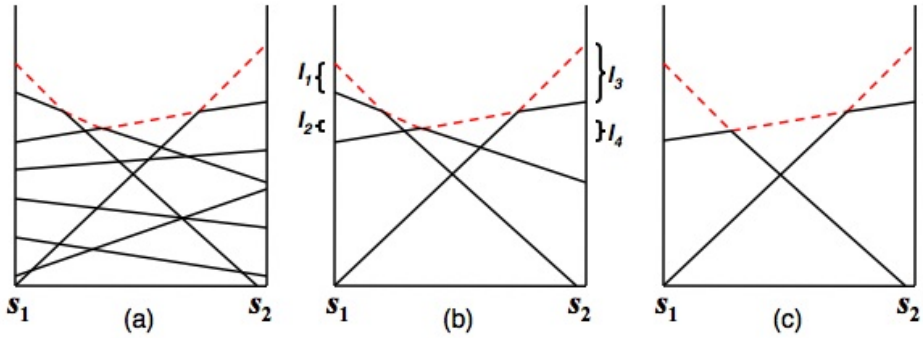

Figure 2: Example sets of vectors for an agent at a given step with dashed lines for the max values (a) before pruning (b) after pruning and maximum loss, l_i, labeled for removing each vector, i, while retaining the other three, and (c) after ϵ-pruning with $l_4 > \epsilon \geq l_2$

Given a vector to be pruned, α and an undominated set U
For variables: $b(s,q)$ and δ
Maximize δ
Such that:

$$\forall i \sum_{s,q} b(s,q)\left[\alpha(s,q) - U_i(s,q)\right] \geq \delta$$

And probability constraint:

$$\sum_{s,q} b(s,q) = 1$$

Table 1: The two agent linear program for pruning a single vector, α, when compared to the current undominated set, U. $b(s,q)$ represents the probability of state s and policy q for the other agent. $\alpha(s,q)$ and $U_i(s,q)$ represent values of these vectors given state s and policy q for the other agent.

PROOF. (sketch)
We will prove this for the two agent problem, but the proof generalizes to any number of agents. For any distribution over states and the policies of the other agent $b(s,q)$ there is some policy, p, that achieves the optimal value. Using ϵ-pruning, if p is not pruned, then the proof is trivial, as the optimal value has not been lost. If p is pruned, we have from Table 1 that for all belief states, and policies, i, in the undominated set

$$\sum_{s,q} b(s,q)\left[\alpha(s,q) - U_i(s,q)\right] \leq \epsilon$$

We substitute ϵ for δ due to the fact that the linear program could not find a belief state that produced a δ greater than ϵ. Thus for all belief states, the left hand side is less than ϵ. We know that U_i was in the undominated set and was not pruned. Therefore the lefthand side of the equation represents a bound on the reduction of joint value.

From the perspective of the other agent, any distribution over the first agent's policies can be replaced with the policies that ϵ-dominate them and this again results in at most a loss in value of ϵ. That is, for any belief over system states and the policies of first agent, $\sum_{s,p} b(s,p)V(p,q,s) \leq \sum_{s,p} b(s,p)\left[V(p',q,s) + \epsilon\right] = \sum_{s,p} b(s,p)V(p',q,s) + \epsilon$ where p' is either the policy p or the policy that is at most ϵ less than p at the given belief over s and q. Thus, from the perspective of the system as a while, one step of ϵ-pruning results in a loss in value of at most ϵ at any belief state, as the agents share a common payoff. □

It follows that one step of e-pruning for each agent, assuming two agents, reduces the joint value by at most 2ϵ. As pruning is described above, it may take several iterations before no agent is able to further prune its undominated set. Because of this, error will accumulate each time ϵ-pruning is done, resulting in a total error of $n\epsilon$ if n pruning steps were necessary on a given step of the dynamic programming.

This error also accumulates over the horizon of the problem. That is, if n_1 ϵ-pruning steps are used on step one of the dynamic programming and n_2 steps are used on step two of the dynamic programming, the total error at the second step is additive, resulting in $n_1\epsilon + n_2\epsilon$. In general, if ϵ-pruning is allowed to continue until convergence on each step, the total error bound for horizon T would be $T(n_1 + n_2 + \ldots + n_T)\epsilon$. This can be seen by noticing that after a backup, the value of any resulting trees is within ϵ of the optimal value for any state. That is,

$$V(p,q,s) \geq R(s,a_p,a_q) + \sum_{s'} P(s'|a_p,a_q,s)$$

$$\sum_{o_1,o_2} O(o_1,o_2|s',a_p,a_q)\left[V(p^*_{o_1},q^*_{o_2},s') - \epsilon\right] =$$

$$R(s,a_p,a_q) + \sum_{s'} P(s'|a_p,a_q,s) \sum_{o_1,o_2} O(o_1,o_2|s',a_p,a_q)$$

$$V(p^*_{o_1},q^*_{o_2},s') - \epsilon$$

where $p^*_{o_1}$ and $q^*_{o_2}$ are the optimal subtrees that may have been pruned on the previous step. This also holds for distributions over the agents' policies.

Given DEC-POMDP, horizon T and bound ϵ
Until horizon T is reached
backup each agent
ϵ-prune each agent once
prune each agent until convergence

Table 2: ϵ-pruning for with a total error bound of $2T\epsilon$.

6. EXPERIMENTS

In this section, we present the results of several variations of bounded dynamic programming algorithms on two DEC-POMDP benchmark problems. We note that two parameters affect the success of the algorithm. The first is the selection of the ϵ value. If ϵ is too high, only one policy tree will be left unpruned. The algorithm will run very quickly, but its accumulated value is likely to be small. If ϵ is too low, then our algorithm becomes the same as Hansen et al., and faces difficulty finding solutions to larger problems and horizons. The second parameter of concern is the number of iterations of ϵ-pruning. We can choose to run ϵ-pruning until no more policies can be removed, but the possible error using this method is high. Alternatively, we can choose to run ϵ-pruning only once per horizon, followed by the regular pruning of Hansen et al. We experimented with both cases and present the results.

The first algorithm is described in Table 2. This approach ϵ-prunes for each agent once per step and then prunes normally until no policy is removed by any agent. Using this algorithm, the resulting solution can be bounded by $2T\epsilon$. The second algorithm is similar to the first, but ϵ-pruning is performed each step until convergence. This results in an error bound that depends on the number of possible policies for the agents for horizon T. Stricter error bound for both methods may also be calculated during execution of the algorithm.

For each of these algorithms, we implemented a binary search over values of ϵ to search for a good value for the given problem. The algorithm is initialized with a given value of ϵ, and it then runs ϵ-pruning on the problem. We refer to this approach as SingleDPSearch if only one step of ϵ-pruning is used for each agent or MultipleDPSearch if ϵ-pruning is used until convergence is reached. If the pruning algorithm does not terminate in a given amount of time (10 minutes per step) with a given amount of memory (2GB), we increase the value of ϵ and try it again. If the pruning algorithm succeeds, then we decrease the value of epsilon and see if we can achieve a higher quality. This may not find the best value of ϵ as larger values of epsilon may not allow the problem to be solvable, while smaller values permit solutions, but it allows for high quality solutions without an exhaustive search. It is also worth noting that this is an anytime algorithm that will quickly return a solution and then increases its quality as time permits.

We also experimented with implementations that start with an epsilon value of zero and raise ϵ as needed. We use a parameter called $MaxTrees$. We proceed with a restriction that we will only backup a certain number of policies, $MaxTrees$, for the next step. If a pruning step results in more policies than $MaxTrees$, we raise ϵ, reprune, and try again, until we are left with less than $MaxTrees$ policies. For our experiments, we chose a $MaxTrees$ value of 30. This reflected the largest number of policies that our experimental workstation could backup in a reasonable amount of time. Because $MaxTrees$ limits the number of policies, this implementation could be applied to problems with larger horizons.

6.1 Broadcast problem

A small DEC-POMDP problem used by Bernstein et al. [1] was a simplified two agent networking example. This problem has 4 states, 2 actions and 5 observations. At each time step, each agent must choose whether or not to send a message. If both agents send, there is a collision and neither gets through. A reward of 1 is given for every step a message is successfully sent over the channel and all other actions receive no reward. Agent 1 has a 0.9 probability of having a message in its queue on each step and agent 2 has only a 0.1 probability. The domain is initialized with only agent 1 possessing a message.

Our results, given in Table 3, show that both versions of DPSearch are able to find high quality solutions. Up to horizon 10, solutions found are the same quality as MBDP Seuken and Zilberstein [6]. At horizon 100 there is a small dropoff. The MaxTrees variant is able to obtain solutions with high value, but the value is a small amount less than the DPSearch algorithms.

Running time was sensitive to the number of policy trees retained in each step. Thus, the MaxTrees algorithm ran in 14.9 seconds for horizon 100. The DPSearches took hours to produce the horizon 100 solution in the table. However, one can note that the running time of DPSearch is somewhat configurable by the user. DPSearch can be terminated at any point, and it will output a solution. If we use the domain's reward function to choose an intial value for ϵ that is guaranteed to overprune, DPSearch will immediately produce a solution that retains one policy for each horizon. It then halves epsilon, until at some point it uses an epsilon that underprunes, resulting in too many policies. After some period of time, depending on when the user configures DPSearch to time out, epsilon is increased, and more policies are pruned. This continues until the user is satisfied with the solution quality.

6.2 Tiger problem

A more challenging domain with 2 states, 3 actions and 2 observations called the multiagent tiger problem was introduced by Nair et al. [5]. 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.

Our results, given in Table 4, show that conducting multiple iterations of ϵ-pruning for each time step is better than performing a single iteration. Both versions of DPSearch found solutions up to horizon 10, at which point the num-

Horizon	SingleDPSearch	MultipleDPSearch	MaxTrees	MBDP	Optimal
3	2.99	2.99	1.90	2.99	2.99
4	3.89	3.89	3.71	3.89	3.89
5	4.79	4.79	4.70	4.79	x
6	5.69	5.69	3.89	5.69	x
7	6.59	6.59	6.16	6.59	x
8	7.49	7.49	7.15	7.49	x
9	8.39	8.39	5.93	8.39	x
10	9.29	9.29	8.51	9.29	x
100	83.41	83.71	73.10	90.29	x

Table 3: Broadcast channel values on different horizons using three different ϵ-pruning algorithms and Seuken and Zilberstein's MBDP. Optimal values are also provided for horizons that can be solved optimally.

ber of trees became intractable for useful values of epsilon. For larger horizons, DPSearch would solve problems by raising epsilon to a large value, and prune every policy except one for each agent. MaxTrees performed worse than DPSearch on very small horizons, but then would catch up for larger horizons. By keeping the number of trees small, it was able find better solutions for some larger horizons than DPSearch. MBDP exceeded the performance of all the epsilon-pruning algorithms that we tried. This is most likely due to the importance of start state information for this problem. While MBDP only retains trees that it finds important for a given start state, our ϵ-pruning methods attempt to retain all trees that may be important for any start state. This makes it more difficult to find concise tree sets for this problem, especially for larger horizons.

Timing was sensitive to the same issues noted in the Broadcast domain. DPSearch would produce its best solution for horizon 100 in a matter of seconds, whereas for horizon 10 it took about an hour. MaxTrees took 1200 seconds for a $MaxTrees$ value of 30. Timing was not overly sensitive to the additional iterations conducted in multipleDPsearch, as opposed to singleDPsearch. For each step, we observed that most pruning was done right away. That is, in multipleDPsearch, agent 0 would prune a large number of policies, then agent 1 would also prune a large number of policies, just like in singleDPsearch. But then when agent 0 would reprune (which is only done in MultipleDPSearch), it would only prune a smaller number of agents in this second pruning iteration, and this second iteration would be much faster. Indeed, although we do not prove a bound on the repruning iterations in this paper, we never observed more than 6 or 7 iterations of repruning for any horizon step.

7. CONCLUSION

In this paper, we showed that we were able to scale up dynamic programming for decentralized POMDPs to arbitrary horizons while maintaining high quality results. These results are produced in an efficient manner that includes a bound on how far the solutions are from the optimal value. By adjusting the number of trees retained in each step of the dynamic programming a trade-off between running time and solution quality may be made. Future directions of this research may include formalizing this tradeoff, so that the agent is wiser in choosing how many policies to retain for the next step.

Our approach is able to obtain solutions to larger problems, which contrasts it to the past pruning technique of Hansen et al. Also, although Seuken and Zilberstein's MBDP is similarly able to solve large problems with higher solution quality, it cannot find a solution within a given bound. MBDP is able to find these solutions by assuming knowledge of the initial state, which it uses for its heuristic search. Such a belief state may not always be available when the agents' policies need to be constructed. It is conceivable that the nature of the domain may be revealed to the agents ahead of time, but the location of the tiger, or the specifics of the initial conditions, may only be revealed at the start of execution, and the agents will already need to have policies ahead of time. Furthermore, it is possible to combine ϵ-pruning with an approach such as MBDP. The MBDP agents would then use ϵ-pruning instead of traditional pruning to prune their search space. This would allow start state information to be used when present and may increase solution quality.

8. REFERENCES

[1] 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*, Edinburgh, Scotland, 2005.

[2] 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*, New York, NY, 2004.

[3] Z. Feng and E. A. Hansen. Approximate planning for factored POMDPs. In *Proceedings of the Sixth European Conference on Planning*, Toledo, Spain, 2001.

[4] 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*, San Jose, CA, 2004.

[5] 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*, Acapulco, Mexico, 2003.

[6] S. Seuken and S. Zilberstein. Memory-bounded dynamic programming for DEC-POMDPs. In *Proceedings of the Twentieth International Joint*

Horizon	SingleDPSearch	MultipleDPSearch	MaxTrees	MBDP	Optimal
1	-2.0	-2.0	-2.0	-2.0	-2.0
2	-4.0	-4.0	-4.0	-4.0	-4.0
3	5.19	5.19	-6.0	5.19	5.19
4	-11.5	-6.7	-21.0	4.80	4.80
5	-22.4	-23.0	-54.0	5.38	x
6	-45.3	-42.6	-56.0	9.91	x
7	-53.0	-66.0	-58.0	9.67	x
8	-81.0	-81.0	-73.0	9.42	x
9	-135.0	-83.0	-88.0	12.57	x
10	-150.0	-98.0	-90.0	13.21	x
50	-750.0	-750.0	-620.0	46.91	x
100	-1500.0	-1500.0	-1500.0	93.24	x

Table 4: Tiger values on different horizons using three different ϵ-pruning algorithms and Seuken and Zilberstein's MBDP. Optimal values are also provided for horizons that can be solved optimally.

Conference on Artificial Intelligence, Hyderabad, India, 2007.

[7] 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.

[8] P. Varakantham, R. Maheswaran, T. Gupta, and M. Tambe. Towards efficient computation of quality bounded solutions in POMDPs. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, Hyderabad, India, 2007.