5-2009

Dynamic Programming Approximations for Partially Observable Stochastic Games

Akshat KUMAR
Singapore Management University, akshatkumarR@smu.edu.sg

Shlomo ZILBERSTEIN
University of Massachusetts Amherst

Dynamic Programming Approximations for Partially Observable Stochastic Games

Akshat Kumar and **Shlomo Zilberstein**
Department of Computer Science
University of Massachusetts
Amherst, MA 01002, USA

Abstract

Partially observable stochastic games (POSGs) provide a rich mathematical framework for planning under uncertainty by a group of agents. However, this modeling advantage comes with a price, namely a high computational cost. Solving POSGs optimally quickly becomes intractable after a few decision cycles. Our main contribution is to provide bounded approximation techniques, which enable us to scale POSG algorithms by several orders of magnitude. We study both the POSG model and its cooperative counterpart, DEC-POMDP. Experiments on a number of problems confirm the scalability of our approach while still providing useful policies.

Introduction

Partially observable stochastic games (POSGs) provide a general, powerful paradigm for decentralized decision making by a group of agents. POSGs allow agents to have conflicting goals in their general form or to share a common reward structure as in DEC-POMDPs (Bernstein et al. 2002). While there has been some progress with applying stochastic games in multi-agent planning and learning (Boutilier 1999), research on stochastic games with partial observability has been sparse. Partial observability is particularly useful when agents cannot completely observe their environment or when different agents perceive different observations. Examples include soccer playing robots, multirobot coordination (Seuken and Zilberstein 2007a) and broadcast channel protocols (Bernstein et al. 2002).

An optimal dynamic programming algorithm for POSGs has been developed by Hansen, Bernstein, and Zilberstein (2004). Dynamic programming for POSGs resembles in some ways POMDP solution techniques, but subtle yet significant differences disallow the direct import of POMDP algorithms. Value iteration for a POMDP works by transforming it into a completely observable continuous-state MDP over belief states. However, in POSGs, the environment state changes as a function of the *joint action* of all agents. Since agents do not have access to other agents policies during plan execution, they cannot maintain the same kind of belief state statistics. This problem is alleviated by introducing the notion of a *generalized belief state* (Hansen, Bernstein, and Zilberstein 2004). That is, agents maintain a belief over the underlying state as well as over the policies of other agents.

The optimal algorithm, however, quickly becomes intractable beyond a small horizon ($\approx$4) (Hansen, Bernstein, and Zilberstein 2004). The main aim of our work is to increase the scalability of dynamic programming algorithms by multiple orders of magnitude. We achieve this by providing a memory bounded representation of the value function defined over the generalized belief space. Our technique also provides an error bound over the optimal policies. Such techniques were shown to be successful in the POMDP domain (Feng and Hansen 2001; Varakantham et al. 2007) and we extend them to POSGs.

Finite horizon cooperative POSGs or DEC-POMDPs have been shown to be NEXP-complete (Bernstein et al. 2002). DEC-POMDP algorithms (Seuken and Zilberstein 2007b) are similar to dynamic programming for general POSGs. Additionally, they exploit the cooperative nature of the agents to introduce heuristics that prune a large part of the search space. We use our approximation techniques in conjunction with an existing approach, MBDP (Seuken and Zilberstein 2007b). MBDP works within a pre-specified memory bounds, but it requires exponential space in the number of agents and number of observations. Our improvement, MBDP-AS, has the *anyspace* characteristic and is much more scalable than MBDP. Experiments on a particularly large problem show the effectiveness of MBDP-AS in finding near optimal policies w.r.t. MBDP, using up to an order of magnitude less space.

Background

This section introduces the POSG model. For details we refer to (Hansen, Bernstein, and Zilberstein 2004).

Partially observable stochastic games

A partially observable stochastic game can be defined as a tuple $\langle Ag, \mathcal{S}, b^0, \{\mathcal{A}_i\}, \{\mathcal{O}_i\}, \mathcal{P}, \{\mathcal{R}_i\}\rangle$

- Ag is a finite set of agents indexed $1, \ldots, n$
- $\mathcal{S}$ is a finite set of environment states
- $b^0 \in \Delta(\mathcal{S})$ is the initial state distribution
- $\mathcal{A}_i$ is the finite set of actions for agent i. The joint action $\overrightarrow{a} \in \times_{i \in Ag} \mathcal{A}_i$ is given by $\langle a_1, \ldots, a_n \rangle$

- $\mathcal{O}_i$ is a finite set of observations for agent i. The joint observation $\overrightarrow{o} \in \times_{i \in Ag} \mathcal{O}_i$ is given by $\langle o_1, \ldots, o_n \rangle$
- $\mathcal{P}$ is a set of state transition and observation probabilities. $\mathcal{P}(s', \overrightarrow{o} | s, \overrightarrow{a})$ is the probability of taking joint action $\overrightarrow{a}$ in state s resulting in transition to state s' and receiving joint observation $\overrightarrow{o}$
- $\mathcal{R}_i : \mathcal{S} \times \overrightarrow{\mathcal{A}} \rightarrow \Re$ is the reward function for agent i, $\overrightarrow{\mathcal{A}}$ is the set of all joint actions.

At each step of the game, the agents simultaneously choose actions, receive a reward and an observation. The environment transitions stochastically into a different state and the above process repeats. The goal for each agent is to maximize the expected sum of rewards it receives during the game. In DEC-POMDPs, the reward function $\mathcal{R}_i$ is the same for all agents requiring them to work cooperatively.

A *policy* for an agent is a mapping from local observation histories to actions. Policies can be represented as trees. In a policy tree, the root node defines the action taken at time step $t = 0$. The edges of the tree represent the possible observations and lead to a new action which will be executed upon receiving that observation. The depth of the tree defines the horizon T of the problem.

Generalized belief state Since agents do not communicate during execution time, they do not directly observe other agents policies and cannot maintain the belief over environment states. A generalized belief is defined for each agent i, which maintains a belief over the underlying state as well as the policies of other agents. It is described by a distribution over $\mathcal{S} \times Q_{-i}$, where Q_{-i} is the set of policies of all other agents. The distribution is denoted by b_i. The generalized belief space for agent i is denoted by $\mathcal{B}_i$. The value of agent i's belief is given by

$$V_i(b_i) = \max_{p \in Q_i} \sum_{s \in \mathcal{S}, q_{-i} \in Q_{-i}} b_i(s, q) V_i(s, p, q_{-i})$$

where Q_i is the set of policies of agent i. Assuming two agents, $V_i(s, p, q)$ is the value of agent i's policy p when other agent executes its policy q and the environment is in state s. It can be easily calculated from problem specifications. Similar to POMDPs, the value function V_i for agent i is $|\mathcal{S} \times Q_{-i}|$ dimensional, piecewise linear and convex.

Dynamic programming for POSGs We describe briefly dynamic programming over generalized belief states. Given depth-t policy trees Q_i^t for each agent i, the multi-agent backup operator H performs the exhaustive backup of these trees to get Q_i^{t+1}. Then it recursively computes the generalized value function V_i^{t+1}. Subsequently, iterated elimination of dominated strategies is performed for each agent i. This step is unique for POSGs and not applicable to POMDPs. The domination condition is

$$\forall b \in \mathcal{B}_i, \exists v_k \in V_i^{t+1} \backslash v_j \; s.t. \; b.v_k \geq b.v_j \qquad (1)$$

The weakly dominated strategies represented by v_j are eliminated from the respective policy tree sets, and this process continues iteratively by alternating over agents until there are no further changes. The above DP algorithm converts the POSG to a normal form representation with reduced set of strategies. Once Q_i^T is computed, standard techniques for selecting equilibria in normal form games can be applied.

Bounded approximation using ϵ pruning

The key issue in the complexity of the DP algorithm is the size of the vector set representing a value function after performing the backup. The size of this vector set is reduced by the iterated elimination of dominated strategies but it still remains a key source of complexity. Our approximation techniques further reduce this vector set size by introducing an error of at most ϵ. The first approximation technique prunes the policies which are higher valued than the remaining policies by at most ϵ. This technique is easy to implement but it is somewhat limited as it does not prune every possible vector allowed by the ϵ error threshold. The next approximation we propose mitigates this shortcoming and reduces the size of the vector set further.

We start by defining the notion of ϵ dominated set followed by the first approximation technique $\mathcal{E}Prune$.

Definition 1. *The vector set Γ representing the value function V is ϵ dominated by the set Γ' representing V' iff $V(b) + \epsilon \geq V'(b)$ for the given ϵ and any b. Γ is ϵ-**parsimonious** if removal of any additional vectors from Γ violates the ϵ domination condition i.e., Γ is the minimal such set.*

We describe below an approximation algorithm $\mathcal{E}Prune$ which returns an ϵ dominated set V corresponding to the set $\mathcal{U}$ produced by the backup operator H.

1. Choose an agent i
2. Initialize the set V_i with the dominant vectors at belief simplex.
3. Choose a vector $u \in \mathcal{U}_i$ according to ordering Θ, use the linear program shown in Algorithm 1 to find if u ϵ-dominates V_i at any belief point b. If not, discard u.
4. Compute u', the best vector at b, remove it from $\mathcal{U}_i$ and add it to V_i. Repeat steps 3, 4 until $\mathcal{U}_i$ is empty.

The above algorithm is run iteratively for each agent i until no more pruning is possible. Each iteration of $\mathcal{E}Prune$ introduces an error of at most ϵ over the set $\mathcal{U}$. For detailed proof we refer to (Varakantham et al. 2007).

Proposition 1. *For a given ordering Θ of the vectors in the set $\mathcal{U}$, $\mathcal{E}Prune$ is not guaranteed to return a **parsimonious** ϵ dominated set V (per iteration).*

Proof. The proof is by a counterexample. Figure 1(a) shows a simplified vector set $\mathcal{U}$ defined over two states s_1 and s_2. The individual numbers identify respective vectors. ϵ

Algorithm 1: $\epsilon DOMINATE(\alpha,\ U,\ \epsilon)$

```
variables: δ, b(s) ∀s ∈ S
maximize δ
    Σ_s b(s)[α(s) − u(s)] ≥ δ + ε ∀u ∈ U
    Σ_{s∈S} b(s) = 1
if δ ≥ 0 return b else return null
```

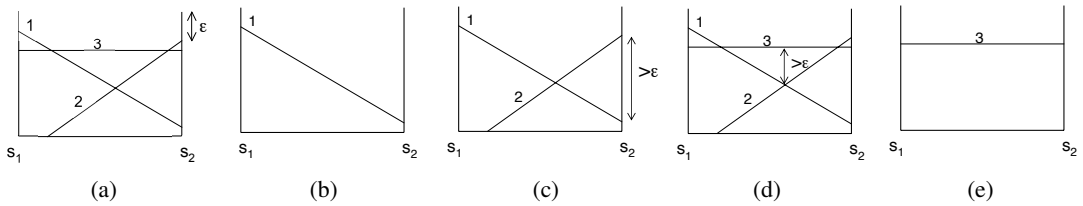

Figure 1: a) shows the vector set $\mathcal{U}$. b) through d) show the steps of algorithm $\mathcal{E}$*Prune*. e) Shows the ϵ dominated parsimonious set for $\mathcal{U}$.

is shown alongside as a line of length ϵ. Consider the ordering $\Theta = \{1, 2, 3\}$ for the vectors in $\mathcal{U}$. During step 1 of $\mathcal{E}$*Prune*, suppose we choose to initialize the set V with the leftmost belief point. The best vector for this corner, 1, is included in the current representation of V (Figure 1(b)). Step 2 of $\mathcal{E}$*Prune* selects the next vector, 2, in Θ. Since 2 dominates the current set V by more than ϵ (Figure 1(c)), it will not be pruned by Algorithm 1. The remaining vector, 3, from $\mathcal{U}$, is considered next and it also dominates current V by more than ϵ (Figure 1(d)), so is retained in V. However we can easily see that V is not parsimonious though ϵ dominated (Figure 1(d)). We can remove vectors 1 and 2 from V without affecting the ϵ domination condition. The ϵ dominated parsimonious set is shown in Figure 1(e). □

An important implication of proposition 1 is that algorithm $\mathcal{E}$*Prune* is sensitive to the ordering Θ used. For a different ordering $\Theta' = \{3, 1, 2\}$ for the set $\mathcal{U}$ in Figure 1(a), we will get the correct parsimonious ϵ-dominated set shown in 1(e). Out next approximation technique improves $\mathcal{E}$*Prune* by further reducing the vector set size keeping the error fixed.

$\mathcal{IE}$*Prune*: improved epsilon pruning

The improved epsilon pruning scheme is based on better use of the ordering of vectors in the set $\mathcal{U}$ for pruning. $\mathcal{IE}$*Prune* prunes much more vectors than $\mathcal{E}$*Prune* while retaining the same error ϵ. For a given upper bound on the size of approximate vector set $\mathcal{V}$, we can pack more *useful* vectors in $\mathcal{V}$, thereby reducing the error ϵ than in $\mathcal{E}$*Prune*.

Algorithm 3 shows the code for $\mathcal{IE}$*Prune*. It takes as input the vector set $\mathcal{U}$, the error threshold ϵ, ordering Θ of vectors in $\mathcal{U}$ and a user defined parameter k. It returns the ϵ dominated set $\mathcal{V}$ for $\mathcal{U}$. $\mathcal{IE}$*Prune* can be logically divided into three phases. Phase 1 (lines 6-10) is similar to $\mathcal{E}$*Prune* which removes all vectors which dominate $\mathcal{V}$ by less than ϵ. In the next two phases, $\mathcal{IE}$*Prune* tries to identify if more vectors can be removed keeping the error fixed. During phase 2, it determines potential group of k vectors which can be removed from V without increasing the error ϵ (using Algorithm 2) and stores them in the $Cliques$ set. In phase 3, it removes all the vectors in the $Cliques$ set from V preemptively, adds back any vector which violates the ϵ domination condition.

Phase 2:. This part forms the basis for Phase 3; it identifies heuristically the potential candidate vector sets which can be further pruned from $\mathcal{V}$. The accuracy of the heuristic depends on the parameter k. The set S denotes this potential set (line 11). It contains all subsets of size k of $\mathcal{V}$, with $|S| = C_k^{|\mathcal{V}|}$. In the block lines 13-15, we try to identify which members of S if removed from $\mathcal{V}$ will still make the set $\mathcal{V}$ ϵ consistent. To ensure this, we also have to take into account the vectors removed in previous iterations (each run of the repeat-until block), stored in the set W (line 13). Set $Cliques$ contains the resulting vectors.

Phase 3: A candidate set is constructed which denotes all vectors that *potentially* can be further pruned (line 16). We preemptively remove all the vectors in candidates set from $\mathcal{V}$ (line 17), and later add any vector that fails the $is\epsilon Consistent$ test ensuring that the error ϵ doesn't increases (line 22).

The above iteration continues until there are no more vectors in $\mathcal{U}$, and the resulting set $\mathcal{V}$ is returned.

Algorithm 2: isϵConsistent(X, $\mathcal{V}$, ϵ)

```
1 forall v ∈ X do
2     if ϵDOMINATE(v, V, ϵ) <> null then
3         Return false
4 Return true
```

Analysis of $\mathcal{IE}$*Prune*

Proposition 2. *$\mathcal{IE}$Prune provides a ϵ dominated set $\mathcal{V}$ over $\mathcal{U}$ i.e. $\forall b \in \mathcal{B}\ \mathcal{V}(b) + \epsilon \geq \mathcal{U}(b)$ where $\mathcal{V}(.)$ and $\mathcal{U}(.)$ represent the respective value function.*

Proof. Phase 1 of $\mathcal{IE}$*Prune* is the same as $\mathcal{E}$*Prune*, so any vector removed at this phase from the set $\mathcal{V}$ is ϵ dominated. For proof see (Varakantham et al. 2007). After this there are two cases possible:

In the first case, $\mathcal{IE}$*Prune* does not prune any additional vectors. Hence the set $\mathcal{V}$ remains ϵ dominated.

In the second case, $\mathcal{IE}$*Prune* removes additional vectors to $\mathcal{E}$*Prune*. Recall that $\mathcal{V}$ represents a value function and $\mathcal{V}(b) = \max_{v \in \Gamma_{\mathcal{V}}} b.v$. $\mathcal{V}$ is the final representation which is obtained at the end of phase 3 for each iteration.

We prove the proposition by contradiction. Suppose that $\exists b \in \mathcal{B}\ s.t.\ \mathcal{V}(b) + \epsilon < \mathcal{U}(b)$. Let $v_b^\epsilon = arg\max_{v \in \Gamma_{\mathcal{V}}} b.v$ and $v_b^\star = arg\max_{v \in \Gamma_{\mathcal{U}}} b.v$ for the current representation of $\mathcal{V}$,$\mathcal{U}$. Also our assumption implies $v_b^\epsilon \neq v_b^\star$ and $v_b^\star \notin \mathcal{V}$. We have

$$v_b^\epsilon.b + \epsilon < v_b^*.b \Rightarrow v_b^*.b > v_b^\epsilon.b + \epsilon \quad (2)$$

Algorithm 3: $\mathcal{IEPrune}$ ($\mathcal{U},\ \epsilon,\ \Theta,\ k$)

1 **Returns:** the set $\mathcal{V}$ ϵ dominated by $\mathcal{U}$
2 $\mathcal{V} \leftarrow$ (remove) Best vectors from $\mathcal{U}$ at belief simplex
3 $W \leftarrow \phi,\ \ Cliques \leftarrow \phi$
4 **repeat**
5 $\quad v \leftarrow \Theta(\mathcal{U})$
6 $\quad b \leftarrow \epsilon$-DOMINATE$(v, \mathcal{V}, \epsilon)$
7 $\quad$ **if** $b = null$ **then**
8 $\quad\quad$ Skip to next iteration
9 $\quad v' \leftarrow$ (remove) Best vector at b from $\mathcal{U}$
10 $\quad \mathcal{V} \leftarrow \mathcal{V} \cup v'$
11 $\quad S \leftarrow \{\{v_i\} | v_i \in \mathcal{V} \wedge |\{v_i\}| = k\}$
12 $\quad$ **forall** $s \in S$ **do**
13 $\quad\quad X \leftarrow s \cup W, \mathcal{V}' \leftarrow \mathcal{V} \backslash s$
14 $\quad\quad$ **if** $is\epsilon Consistent(X,\ \mathcal{V}',\ \epsilon)$ **then**
15 $\quad\quad\quad Cliques \leftarrow Cliques \cup s$
16 $\quad Candidates \leftarrow \cup_i s_i \mid s_i \in Cliques$
17 $\quad \mathcal{V}' \leftarrow \mathcal{V} \backslash Candidates$
18 $\quad Pruned \leftarrow \phi$
19 $\quad$ **forall** $v \in Candidates$ **do**
20 $\quad\quad X \leftarrow \{v\} \cup W$
21 $\quad\quad$ **if** $\neg is\epsilon Consistent(X,\ \mathcal{V}',\ \epsilon)$ **then**
22 $\quad\quad\quad \mathcal{V}' \leftarrow \mathcal{V}' \cup \{v\}$
$\quad\quad$ **else**
23 $\quad\quad\quad Pruned \leftarrow Pruned \cup \{v\}$
24 $\quad \mathcal{V} \leftarrow \mathcal{V}', \quad W \leftarrow W \cup Pruned$
until $\mathcal{U} = \phi$
25 **Return** $\mathcal{V}$

Consider the case when v_b^* was considered by $\mathcal{IEPrune}$. If it was removed during phase 1, then the ϵ domination proof of phase 1 contradicts our assumption. So surely it must have been removed during phase 3. This implies $v_b^\star \in Candidates$ and consequently $v_b^\star$ passed the $is\epsilon Consistent$ test (line 21) to be eligible for pruning. From the condition of LP-DOMINATE in $is\epsilon Consistent$ test we have:

$$\exists \tilde{v} \in \mathcal{V}' \ \ \nexists b \in \mathcal{B}\ s.t.\ v_b^\star.b \geq \delta + \epsilon + \tilde{v}.b \ and\ \delta \geq 0 \quad (3)$$

At any point during the iteration of for loop (20-23) the following holds: $\mathcal{V}' \subseteq \mathcal{V}$ and since v_b^ϵ is the best vector in $\mathcal{V}$ at the belief b, we have $v_b^\epsilon.b \geq \tilde{v}.b$. Combining this with eq 3 and substituting $\delta = 0$, we get $\nexists b \in \mathcal{B}\ s.t.\ v_b^\star.b \geq \epsilon + v_b^\epsilon.b$. This is a contradiction to our assumption in Eq. 2. Hence the proposition must hold. □

It can be shown that the error ϵ accumulates during each invocation of $\mathcal{IEPrune}$ like $\mathcal{EPrune}$ (Varakantham et al. 2007).

POSG experiments

We experiment on *Multi access broadcast channel* (MABC) domain introduced in (Hansen, Bernstein, and Zilberstein 2004) and compare the performance of $\mathcal{EPrune}$, $\mathcal{IEPrune}$ with the optimal dynamic programming. The MABC problem involves two agent which take control of a shared channel for sending messages and try to avoid collisions. Table 1 shows the comparison of optimal dynamic programming with the approximation schemes. The space requirements for optimal DP become infeasible after horizon 4. In our approximation schemes, we bounded the number of policies to 30 per agent and epsilon value was increased until the policy tree sets were within the required size. Table 1 clearly shows the increased scalability of the approximation techniques. Their performance matched with the optimal policy until horizon 4. Using the bounded memory, these schemes could execute up to horizon 100 (possibly even further), multiple orders of magnitude over the optimal algorithm. When comparing with a random policy, even for larger horizons the approximations provided *useful* policies achieving nearly 200% of the random value.

Horizon	*Optimal* # of Policies	*Optimal* Policy Value	$\mathcal{EPrune}$	$\mathcal{IEPrune}$	*Random*
2	$(6, 6)$	2.00	2.00	2.00	1
3	$(20, 20)$	2.99	2.99	2.99	1.49
4	$(300, 300)$	3.89	3.89	3.89	1.99
5	-	-	4.79	4.79	2.47
8	-	-	7.49	7.49	3.93
10	-	-	9.29	9.29	4.90
100	-	-	73.10	82.10	48.39

Table 1: Comparison of the scalability of different approaches over the MABC problem

The next experiment was performed on the larger multi-agent tiger problem (Seuken and Zilberstein 2007b). Figure 2 shows the comparison of $\mathcal{IEPrune}$ and $\mathcal{EPrune}$ on total error and ϵ value used for each horizon. We show the relative percentage of the total error accumulated and the ϵ per horizon i.e. $\epsilon^{\mathcal{IEPrune}}/\epsilon^{\mathcal{EPrune}} \times 100$. The graph clearly shows that $\mathcal{IEPrune}$ prunes more vectors and packs more useful vectors within the given space requirements thereby reducing the total error and ϵ than $\mathcal{EPrune}$. The next section describes application of $\mathcal{IEPrune}$ for DEC-POMDPs resulting in an any space algorithm for solving DEC-POMDPs.

Any-space dynamic programming for DEC-POMDPs

DEC-POMDPs are a special class of POSGs which allow cooperative agents to share a common reward structure. The number of possible policies per agent is of the order $O(|\mathcal{A}|^{|\mathcal{O}|^T})$ for horizon T. This explains why effective pruning is crucial to make algorithms scalable.

Recently, a memory bounded dynamic programming approach MBDP has been proposed for DEC-POMDPs (Seuken and Zilberstein 2007b). MBDP limits the number of policy trees retained at the end of each horizon to a predefined threshold $MaxTrees$. The backup operation produces $|\mathcal{A}|MaxTrees^{|\mathcal{O}|}$ policies for each agent for the next iteration. MBDP then uses a portfolio of top-down heuristics to select $MaxTrees$ belief points and retains only the best

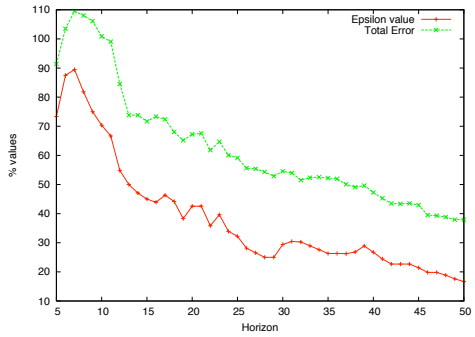

Figure 2: $\mathcal{IE}Prune$ versus $\mathcal{E}Prune$: % total error and ϵ value used for each horizon

Algorithm 4: MBDP-Prune($\{Q_i\}$, $MaxJointPolicies$)

```
1 JointPolicies ← |Q1||Q2|
2 repeat
3     forall agent i ∈ Ag do
4         Size_i ← MaxJointPolicies/|Q_-i|
5         Q'_i ← Randomly select Size_i policies from Q_i
6         evaluate(Q'_i, Q_-i)
7         IEPrune(V_Q'i, ε, Θ, k)
8         IncreaseEpsilon?
9         JointPolicies ← |Q1||Q2|
  until JointPolicies≤ MaxJointPolicies
```

policies for each agent per sampled belief. To further reduce the complexity (Carlin and Zilberstein 2008) use partial backups in their approach MBDP-OC and limit the number of possible observations to a constant $MaxObs$. This results in $|\mathcal{A}|MaxTrees^{MaxObs}$ policies for the next iteration per agent. MBDP provides significant speedups over existing algorithms but has *exponential* space complexity in the observation space. We use the improved epsilon pruning scheme $\mathcal{IE}Prune$ to make MBDP any space: MBDP-AS, and also provide a bound on the error produced by $\mathcal{IE}Prune$ over MBDP.

MBDP-AS

MBDP-AS takes a user defined parameter $MaxJointPolicies$ which restricts MBDP-AS to store only $MaxJointPolicies$ joint policies, effectively reducing the space requirements from $|\mathcal{A}|^2MaxTrees^{2\cdot MaxObs}$ to $|S|MaxJointPolicies$ for two agents. MBDP-AS introduces an extra step of pruning shown in Algorithm 4 just after the backup operation. Algorithm 4 prunes policies of each agent until the total joint policies is within the limit $MaxJointPolicies$. After this pruning step all agents evaluate their policies against each other requiring only $O(|S|MaxJointPolicies)$ space and retain the best policies for the beliefs sampled by the top-down heuristic. We describe the Algorithm 4 in the following.

The motivation behind Algorithm 4 is that many policy vectors which provide *similar rewards* can be pruned while retaining only a representative subset of them. Such systematic pruning requirement is ideally achieved with $\mathcal{IE}Prune$. We call each execution of pruning (lines 4-9) an iteration of MBDP-Prune. To avoid evaluating every joint policy for agent i, we select a random subset of i's policies such that current total joint policies are within $MaxJointPolicies$ limit (lines 4,5). These joint policies are then evaluated (line 6). For the $|S \times Q_{-i}|$ dimensional multi-agent belief b, the policy vector associated with agent i's policy p is given by $V(p) = (\ldots, V(p, q_{-i}, s)\ldots), q_{-i} \in Q_{-i}, s \in \mathcal{S}$. The set of all policy vectors of agent i is passed to $\mathcal{IE}Prune$ (line 7) which gives a smaller ϵ dominated set. For all the vectors which are pruned by $\mathcal{IE}Prune$ we also prune the corresponding policies from Q_i. We then decide if to increase ϵ (line 8) to aggravate the pruning considering the time overhead, and if no policy was pruned in the last iteration. This iterative pruning continues until the $JointPolicies$ are within the specified limit.

Analysis of the MBDP-Prune operation

Proposition 3. *The maximum number of iterations required by the MBDP-Prune operation is $(\Delta - \epsilon)/\delta$, where ϵ is the initial epsilon value used, δ is the increment in epsilon after each iteration. $\Delta = \max_{b \in \mathcal{B}_M} \left(\min_{V_i \in \mathcal{V}_{Q_i}} (V_b^\star.b - V_i.b)\right)$ and $V_b^\star = argmax_{V_i \in \mathcal{V}_{Q_i}} V_i.b$ for any b.*

Proof. The MBDP-Prune operation will continue to increase epsilon (line 8) by the amount δ until the current joint policies are within $MaxJointPolicies$ limit. The worst case is when there is only one policy vector V' allowed. V' must dominate all other vectors at some multi-agent belief i.e. $\exists b \; s.t. \; (V'.b - V_i.b) \geq 0 \; \forall V_i \in \mathcal{V}_{Q_i}, V' \neq V_i$. The quantity $(V'.b - V_i.b)$ must be the maximum for all belief points so that setting epsilon to that value requires $\mathcal{IE}Prune$ to prune every vector other than V'. This amount is given by Δ. Consequently, the number of iterations required to raise epsilon to Δ are $(\Delta - \epsilon)/\delta$. □

Proposition 4. *Each iteration of MBDP-Prune results in joint policy values that are at most ϵ below the optimal joint policy value in MBDP for any belief state.*

This results directly from proposition 2. For further details we refer the reader to (Amato, Carlin, and Zilberstein 2007). Proposition 4 forms the basis of quality guarantees over the approximation. If n iterations of MBDP-Prune are required to bring down the joint policies within the limit, the *maximum error* at any belief is bounded by $n\epsilon$ over MBDP with no pruning.

MBDP-AS experiments

We built MBDP-AS on top of MBDP-OC (Carlin and Zilberstein 2008) which is similar to MBDP except that it uses partial backups. The test problem we use is cooperative box pushing (Carlin and Zilberstein 2008), which is substantially larger than other DEC-POMDP benchmarks such as Tiger and Multiagent Broadcast Channel (Seuken and Zilberstein 2007b). In this domain, two agents are required to push boxes into the goal area. Agents have 4 available actions and

horizon	*Joint Policies*=11664 IMBDP	MBDP-OC	*Joint Policies*=2500 MBDP-AS
5	79	72	69
10	91	103	93
20	96	149	148
30	89	168	170
40	68	244	228
50	81	278	268

Table 2: Comparison of IMBDP, MBDP-OC, MBDP-AS on the Repeated Box Pushing domain with various horizons (with *MaxTrees* = 3, *MaxObs* = 3)

horizon	*Joint Policies*=2500 MaxTree = 3	*Joint Policies*=7000 MaxTree = 4
5	69	77
10	93	105
20	148	150
40	228	248

Table 3: Comparison of MBDP-AS for two settings: *MaxTree=3* and *MaxTree = 4*. *MaxObs=3* in both cases

can receive 5 possible observations. The number of states is 100. Full backups in MBDP are costly, so IMBDP (Seuken and Zilberstein 2007a) and MBDP-OC (Carlin and Zilberstein 2008) take partial backups with $MaxObs$ observations. We compare all three algorithms IMBDP, MBDP-OC and MBDP-AS on two metrics: space requirements, which is shown by the total number of joint policies stored and the best policy value.

Table 2 compares all three algorithms on these metrics. MBDP-OC and IMBDP evaluate and store all 11664 joint policies, whereas in MBDP-AS we set the $MaxJointPolicies = 2500$, which is nearly one fifth of the total policies. Even at this greatly reduced policies the quality of policies in MBDP-AS is comparable to MBDP-OC. We only lose slightly for higher horizons (40-50) on MBDP-OC while always superior to IMBDP for nearly all horizons. This clearly depicts the effectiveness of MBDP-AS which prunes selectively using $\mathcal{IE}Prune$ algorithm.

In the next set of experiments, $MaxTrees$ was increased to 4. This requires both IMBDP and MBDP-OC to store and evaluate 65536 policies and is infeasible due to excessive memory requirements. MBDP-AS still scaled well using $MaxJointPolicies = 7000$, about an order of magnitude less policies than total joint policies. Table 3 compares the result for MBDP-AS with $MaxTrees = 3$ and $MaxTrees = 4$. As expected the quality of solution increased for all horizons with the number of MaxTrees. This further confirms the scalability of our approach and effectiveness of MBDP-Prune operation.

As far as execution time is concerned MBDP-AS provided interesting results. The average time per horizon for $MaxTrees = 3$ and $MaxObs = 3$ for MBDP-OC was 470sec. For MBDP-AS with the same parameters the average execution time per horizon was 390sec. The overhead of linear programs used for pruning was more than compensated by the lesser policy evaluations which MBDP-AS performed. Total policy evaluations in MBDP-OC per horizon was $2 \times 11664 = 23328$. MBDP-AS on the contrary evaluates only $MaxJointPolicies$ at one time and prunes many policies. The effort to evaluate pruned policies in the next iteration is saved, leading MBDP-AS to evaluate 16300 policies on average.

Conclusion

Our work targets the dynamic programming bottleneck which is a common problem associated with value iteration in DEC-POMDPs and POSGs. We have improved an existing epsilon pruning based approximation technique and our improvement provides better error bound across all horizons on the examined tiger problem. As a second application of our new pruning technique, we incorporate it into an existing algorithm, MBDP, for DEC-POMDPs and address a major weakness of it: exponential space complexity. The new algorithm MBDP-AS is an any-space algorithm, which also provides error bounds on the approximation. Experiments confirm its scalability even on problem sizes where other algorithms failed. These results contribute to the scalability of a range of algorithms that rely on epsilon pruning to approximate the value function.

Acknowledgments

This work was supported in part by the AFOSR under Grant No. FA9550-08-1-0181 and by the NSF under Grant No. IIS-0812149.

References

Amato, C.; Carlin, A.; and Zilberstein, S. 2007. Bounded dynamic programming for Decentralized POMDPs. In *Proc. of AAMAS workshop on Multi-agent sequential decision making in uncertain domains*, 31–45.

Bernstein, D.; Givan, R.; Immerman, N.; and Zilberstein, S. 2002. The complexity of decentralized control of Markov Decision Processes. *Mathematics of Operations Research* 27:819–840.

Boutilier, C. 1999. Sequential optimality and coordination in multiagent systems. In *Proc. of International Joint Conference on Artificial Intelligence*, 478–485.

Carlin, A., and Zilberstein, S. 2008. Value-based observation compression for DEC-POMDPs. In *Proc. of International Conference on Autonomous Agents and Multiagent Systems*, 501–508.

Feng, Z., and Hansen, E. 2001. Approximate planning for factored POMDPs. In *Proc. of European Conference on Planning*.

Hansen, E. A.; Bernstein, D. S.; and Zilberstein, S. 2004. Dynamic programming for partially observable stochastic games. In *Proc. of AAAI*, 709–715.

Seuken, S., and Zilberstein, S. 2007a. Improved memory-bounded dynamic programming for decentralized POMDPs. In *Proc. of Conference on Uncertainty in Artificial Intelligence*.

Seuken, S., and Zilberstein, S. 2007b. Memory-bounded dynamic programming for DEC-POMDPs. In *Proc. of International Joint Conference on Artificial Intelligence*, 2009–2015.

Varakantham, P.; Maheswaran, R.; Gupta, T.; and Tambe, M. 2007. Towards efficient computation of error bounded solutions in POMDPs: Expected Value Approximation and Dynamic Disjunctive Beliefs. In *Proc. of International Joint Conference on Artificial Intelligence*, 2638-2644.