Efficient Maximization in Solving POMDPs

Zhengzhu Feng
Computer Science Department
University of Massachusetts
Amherst, MA 01003
fengzz@cs.umass.edu

Shlomo Zilberstein
Computer Science Department
University of Massachusetts
Amherst, MA 01003
shlomo@cs.umass.edu

Abstract

We present a simple, yet effective improvement to the dynamic programming algorithm for solving partially observable Markov decision processes. The technique targets the vector pruning operation during the maximization step, a key source of complexity in POMDP algorithms. We identify two types of structures in the belief space and exploit them to reduce significantly the number of constraints in the linear programs used for pruning. The benefits of the new technique are evaluated both analytically and experimentally, showing that it can lead to significant performance improvement. The results open up new research opportunities to enhance the performance and scalability of several POMDP algorithms.

Introduction

A partially observable Markov decision process (POMDP) models an agent acting in an uncertain environment, equipped with imperfect actuators and noisy sensors. It provides an elegant and expressive framework for modeling a wide range of problems in decision making under uncertainty. However, this expressiveness in modeling comes with a prohibitive computational cost when it comes to solving a POMDP and obtaining an optimal policy. Improving the scalability of solution methods for POMDPs is thus a critical research topic.

Standard solution methods for POMDPs rely on performing a dynamic programming update of the value function, represented by a finite set of linear vectors over the state space. A key source of complexity is the size of the value function representation, which grows exponentially with the number of observations. Fortunately, a large number of vectors in this representation can be pruned away without affecting the values using a linear programming (LP) method. Solving the resulting linear programs is therefore the main computation in the DP update.

Consequently, many research efforts have focused on improving the efficiency of vector pruning. The pruning happens at three stages of the DP update, namely the projection stage, the cross-sum stage, and the maximization stage. During the cross-sum stage, the number of vectors increases exponentially, making it the major bottle-neck in the DP update process. As a result, most research efforts focus on the cross-sum stage. The incremental pruning (IP) algorithm (Zhang & Liu 1996; Cassandra, Littman, & Zhang 1997) was designed to address this problem by interleaving the cross-sum and the pruning which leads to significantly reduced number of linear programs to be solved. Recently, we developed a region-based variant of IP that can exploit the local structure of the belief space to reduce the size of the linear programs, leading to exponential improvement of the cross-sum stage (Feng & Zilberstein 2004). However, these algorithms do not address the performance of the maximization stage that prunes the combination of the results from the cross-sum stage, which can be exponentially large as well. Therefore the improvement to the overall DP update process by these methods is limited.

In this paper, we identify certain properties of the projection and maximization stages and show how they can be exploited to greatly accelerate the DP process. We build on the region-based cross-sum pruning approach we previously developed, and specifically address here the maximization step. We show that in the maximization stage, only vectors whose witness regions are close to each other in the belief space are needed for testing dominance. We show how this closeness information can be obtained during the cross-sum stage at little cost. Although this method leaves some dominated vectors undetected, we show that typical reachability and observability structure in a problem allows such dominated vectors to be pruned efficiently in a subsequent projection pruning stage. Combining these two ideas, our algorithm can deliver a significant performance improvement to the whole dynamic programming algorithm.

The POMDP Model

We consider a discrete time POMDP defined by the tuple (S, A, P, R, Z, O), where

- S is a finite set of states;
- A is a finite set of actions.
- P is the transition model, $P^a(s'|s)$ is the probability of reaching state s' if action a is taken in state s;
- R is the reward model, $R^a(s)$ is the expected immediate reward for taking action a in state s;
- Z is a finite set of observations that the agent can sense;

- O is the observation model, $O^a(z|s')$ is the probability of observing z if action a is taken and resulted in state s'.

We are interested in maximizing the infinite horizon total discounted reward, where $\beta \in [0, 1)$ is the discount factor. The standard approach to solving a POMDP is to convert it to a *belief-state* MDP. A belief state b is a probability distribution over the state space $b : S \rightarrow [0, 1]$, such that $\sum_{s \in S} b(s) = 1.0$. Given a belief state b, representing the agent's current estimate of the underlying states, the next belief state b' is the revised estimate as a result of taking action a and receiving observation z. It can be computed using Bayesian conditioning as follows:

$$b'(s') = \frac{1}{P^a(z|b)} O^a(z|s') \sum_{s \in S} P^a(s'|s) b(s),$$

where $P^a(z|b)$ is a normalizing factor:

$$P^a(z|b) = \sum_{s' \in S} \left[O^a(z|s') \sum_{s \in S} P^a(s'|s) b(s) \right]$$

We use $b' = T_a^z(b)$ to refer to belief update. It has been shown that a belief state updated this way is a sufficient statistic that summarizes the entire history of the process. It is the only information needed to perform optimally. An equivalent, completely observable MDP, can be defined over this belief state space as the tuple $(\mathcal{B}, A, T, R_{\mathcal{B}})$, where $\mathcal{B}$ is the infinite space of all belief states, A is the action set as before, T is the belief transition function as defined above, and $R_{\mathcal{B}}$ is the reward model, constructed from the POMDP model as follows: $R^a_{\mathcal{B}}(b) = \sum_{s \in S} b(s) R^a(s)$.

In this form, a POMDP can be solved by iteration of a *dynamic programming update* that improves a value function $V : \mathcal{B} \rightarrow \Re$. For all belief states $b \in \mathcal{B}$:

$$V'(b) = \max_{a \in A} \left\{ R^a_{\mathcal{B}}(b) + \beta \sum_{z \in Z} P^a(z|b) V(T(b)) \right\}. \quad (1)$$

Performing the DP update is challenging because the space of belief states is continuous. However, Smallwood and Sondik (Smallwood & Sondik 1973) proved that the DP backup preserves the piecewise linearity and convexity of the value function, leading the way to designing POMDP algorithms. A piecewise linear and convex value function V can be represented by a finite set of $|S|$-dimensional vectors of real numbers, $\mathcal{V} = \{v^0, v^1, \ldots, v^k\}$, such that the value of each belief state b is defined by $V(b) = \max_{v^i \in \mathcal{V}} b \cdot v^i$, where $b \cdot v := \sum_{s \in S} b(s) v(s)$ is the "dot product" between a belief state and a vector. Moreover, a piecewise linear and convex value function has a unique minimal-size set of vectors that represents it. This representation of the value function allows the DP update to be computed exactly.

Vector Pruning in Dynamic Programming

Note that the computation of V' in Equation (1) can be divided into three stages: (Cassandra, Littman, & Zhang 1997)

$$V^{a,z}(b) = \frac{R^a_{\mathcal{B}}(b)}{|Z|} + \beta P^a(z|b) V(T(b)) \quad (2)$$

$$V^a(b) = \sum_{z \in Z} V^{a,z}(b) \quad (3)$$

$$V'(b) = \max_{a \in A} V^a(b) \quad (4)$$

Each of these value functions is piecewise linear and convex, and can be represented by a unique minimum-size set of vectors. We use the symbols $\mathcal{V}'$, $\mathcal{V}^a$, and $\mathcal{V}^{a,z}$ to refer to these minimum-size sets.

Using the script letters $\mathcal{U}$ and $\mathcal{W}$ to denote sets of vectors, we adopt the following notation to refer to operations on sets of vectors. The *cross-sum* of two sets of vectors, $\mathcal{U}$ and $\mathcal{W}$, is defined by $\mathcal{U} \oplus \mathcal{W} = \{u + w | u \in \mathcal{U}, w \in \mathcal{W}\}$. An operator that takes a set of vectors $\mathcal{U}$ and reduces it to its unique minimum form is denoted $\mathbb{PR}(\mathcal{U})$. We also use $\mathbb{PR}(\mathcal{U})$ to denote the resulting minimum set. Formally, $u \in \mathbb{PR}(\mathcal{U})$ if and only if $u \in \mathcal{U}$, and $\exists b \in \mathcal{B}$ such that for $\forall u' \neq u \in \mathcal{U}$, $u \cdot b > u' \cdot b$. Using this notation, the three stages of computation can be expressed as follows:

$$\mathcal{V}^{a,z} = \mathbb{PR}\left(\{v^{a,z,i} | v^i \in \mathcal{V}\}\right), \quad (5)$$

$$\mathcal{V}^a = \mathbb{PR}\left(\oplus_{z \in Z} \mathcal{V}^{a,z}\right) \quad (6)$$

$$\mathcal{V}' = \mathbb{PR}\left(\cup_{a \in A} \mathcal{V}^a\right) \quad (7)$$

where $v^{a,z,i}$ is a *projection* from v^i computed by

$$v^{a,z,i}(s) = \frac{R^a(s)}{|Z|} + \beta \sum_{s' \in S} O^a(z|s') P^a(s'|s) v^i(s'). \quad (8)$$

We refer to these three stages as *projection pruning*, *cross-sum pruning*, and *maximization pruning*, respectively. Table 1 summarizes an algorithm from (White 1991) that reduces a set of vectors to a unique, minimal-size set by removing "dominated" vectors, that is, vectors that can be removed without affecting the value of any belief state.

There are two standard tests for dominated vectors. The simplest method is to remove any vector u that is pointwise dominated by another vector w. That is, $u(s) \leq w(s)$ for all $s \in S$. The procedure POINTWISE-DOMINATE in Table 1 performs this operation. Although this method of detecting dominated vectors is fast, it can only remove a small number of dominated vectors.

There is a linear programming method that can detect all dominated vectors. The main algorithm is summarized in the procedure $\mathbb{PR}(\mathcal{W})$ in Table 1. Given a set of vectors $\mathcal{W}$, it extracts non-dominated vectors from $\mathcal{W}$ and puts them in the set $\mathcal{D}$. Each time a vector w is picked from $\mathcal{W}$, it is tested against $\mathcal{D}$ using the linear program listed in procedure LP-DOMINATE. The linear program determines whether adding w to $\mathcal{D}$ improves the value function represented by $\mathcal{D}$ for any belief state b. If it does, the vector in $\mathcal{W}$ that gives the maximal value at belief state b is extracted from $\mathcal{W}$ using the procedure BEST, and is added to $\mathcal{D}$. Otherwise w is a dominated vector and is discarded. The symbol $<_{lex}$ in procedure BEST denotes lexicographic ordering. Its significance in implementing this algorithm was elucidated by Littman (1994).

Note that using this algorithm to prune a set $\mathcal{W}$, the number of constraints in each linear program is bounded by $|\mathbb{PR}(\mathcal{W})|$, the size of the resulting set. In the worst case,

procedure POINTWISE-DOMINATE($w, \mathcal{U}$)
1. for each $u \in \mathcal{U}$
2. if $w(s) \leq u(s),\ \forall s \in S$ then return true
3. return false

procedure LP-DOMINATE($w, \mathcal{U}$)
4. solve the following linear program
   variables: $d,\ b(s)\ \forall s \in S$
   maximize d
   subject to the constraints
   $b \cdot (w - u) \geq d,\ \forall u \in \mathcal{U}$
   $\sum_{s \in S} b(s) = 1$
5. if $d \geq 0$ then return b
6. else return nil

procedure BEST($b, \mathcal{U}$)
7. $max \leftarrow -\infty$
8. for each $u \in \mathcal{U}$
9. if ($b \cdot u > max$) or (($b \cdot u = max$) and ($u <_{lex} w$))
10. $w \leftarrow u$
11. $max \leftarrow b \cdot u$
12. return w

procedure $\mathbb{PR}(\mathcal{W})$
13. $\mathcal{D} \leftarrow \emptyset$
14. while $\mathcal{W} \neq \emptyset$
15. $w \leftarrow$ any element in $\mathcal{W}$
16. if POINTWISE-DOMINATE($w, \mathcal{D}$) = true
17. $\mathcal{W} \leftarrow \mathcal{W} - \{w\}$
18. else
19. $b \leftarrow$ LP-DOMINATE($w, \mathcal{D}$)
20. if $b = nil$ then
21. $\mathcal{W} \leftarrow \mathcal{W} - \{w\}$
22. else
23. $w \leftarrow$ BEST($b, \mathcal{W}$)
24. $\mathcal{D} \leftarrow \mathcal{D} \cup \{w\}$
25. $\mathcal{W} \leftarrow \mathcal{W} - \{w\}$
26. return $\mathcal{D}$

Table 1: Algorithm for pruning a set of vectors $\mathcal{W}$.

$|\mathbb{PR}(\mathcal{W})|$ can be as large as $|\mathcal{W}|$. With this in mind, let's examine the number of constraints in the linear programs during the three pruning stages:

Projection pruning Given the input value function $\mathcal{V}$, the linear programs in the projection pruning (Eq. 5) have worst case number of constraints of $|\mathcal{V}^{a,z}|$. In the worst case, $|\mathcal{V}^{a,z}| = |\mathcal{V}|$. However, for many practical domains, $\mathcal{V}^{a,z}$ is usually much smaller than $\mathcal{V}$. In particular, a problem usually exhibits the following local structure:

- **Reachability**: from state s, only a limited number of states s' can be reachable through action a.
- **Observability**: for observation z, there are only a limited number of states in which z is observable after action a is taken.

As a result, the belief update for a particular (a, z) pair usually maps the whole belief space $\mathcal{B}$ into a small subset $T_a^z(\mathcal{B})$. Effectively, only values of $\mathcal{V}$ over this belief subset need to be backed up in Equation 8. The number of vectors needed to represent $\mathcal{V}$ over the subset can be much smaller, and the projection pruning can in fact be seen as a way of finding the minimal subset of $\mathcal{V}$ that represents the same value function over $T_a^z(\mathcal{B})$. We will exploit this fact in our algorithm, by shifting some of the pruning in the maximization stage to the projection stage of the next DP update.

Cross-sum pruning The cross-sum is the source of the exponential growth of the value function, since $|\oplus_z \mathcal{V}^{a,z}| = \prod_z |\mathcal{V}^{a,z}|$. Using the standard pruning algorithm, there are $\prod_z |\mathcal{V}^{a,z}|$ linear programs to be solved, and the the number of constraints in these linear programs can be as large as $|\mathcal{V}^a|$. The incremental pruning algorithm (Cassandra, Littman, & Zhang 1997) aims at reducing the number of linear programs that need to be solved in the cross-sum pruning stage, by interleaving the pruning and the cross-sum operators:

$$\mathcal{V}^a = \mathbb{PR}(\mathcal{V}^{a,z_1} \oplus \mathbb{PR}(\mathcal{V}^{a,z_2} \oplus \cdots \mathbb{PR}(\mathcal{V}^{a,z_{k-1}} \oplus \mathcal{V}^{a,z_k}) \cdots))$$

This greatly reduces the number of linear programs. However, although in practice there are usually a large number of vectors that are dominated in $\oplus_z \mathcal{V}^{a,z}$, the size of $\mathcal{V}^a$ still represents an exponential increase over the size of the inputs. Therefore in incremental pruning, each linear program can still be very large. Recently, Feng and Zilberstein (2004) introduced an improved version of incremental pruning that reduced the worst case number of constraints to $\sum_z |\mathcal{V}^{a,z}|$, leading to an exponential speed-up of the cross-sum stage.

Maximization pruning The maximization pruning presents yet another bottleneck in the DP process, since it needs to prune the union of the cross-sum value functions for all actions, and each cross-sum $\mathcal{V}^a$ can be exponential in the size of the previous value function $\mathcal{V}$. In this paper, we propose a simple algorithm for selecting constraints for the linear programs used in the maximization pruning stage. We borrow the region-based view presented in (Feng & Zilberstein 2004), and pick constraints for use in the maximization pruning linear program according to the local belief space structure.

Region-Based Cross-Sum Pruning

In this section, we review the region-based pruning algorithm for the cross-sum stage in (Feng & Zilberstein 2004), and introduce the notation used by our algorithm. Recall that each vector $u \in \mathcal{U}$ defines a *witness region* $\mathcal{B}_\mathcal{U}^u$ over which u dominates all other vectors in $\mathcal{U}$ (Littman, Cassandra, & Kaelbling 1996):

$$\mathcal{B}_\mathcal{U}^u = \{b | b \cdot (u - u') > 0, \forall u' \in \mathcal{U} - \{u\}\}. \qquad (9)$$

Note that each inequality in Equation (9) can be represented by a vector, $(u - u')$, over the state space. We call the inequality associated with such a vector a *region constraint*, and use the notation $\mathbb{L}(\mathcal{B}_\mathcal{U}^u) := \{(u - u') | u' \in \mathcal{U} - \{u\}\}$ to represent the set of region constraints defining $\mathcal{B}_\mathcal{U}^u$. Note that for any two regions $\mathcal{B}_\mathcal{U}^u$ and $\mathcal{B}_\mathcal{W}^w$,

$$\mathbb{L}(\mathcal{B}_\mathcal{U}^u \cap \mathcal{B}_\mathcal{W}^w) = \mathbb{L}(\mathcal{B}_\mathcal{U}^u) \cup \mathbb{L}(\mathcal{B}_\mathcal{W}^w). \qquad (10)$$

Recall that the cross-sum stage performs the following pruning: $\mathcal{V}^a = \mathbb{PR}(\mathcal{V}^{a,z_1} \oplus \mathcal{V}^{a,z_2} \oplus \cdots \oplus \mathcal{V}^{a,z_k})$. We use

$$v_1 + \cdots + v_k \in (\mathcal{V}^{a,z_1} \oplus \cdots \oplus \mathcal{V}^{a,z_k})$$

to refer to a vector in the cross-sum, implying $v_i \in \mathcal{V}^{a,z_i}$. It can be shown that $\sum_i v_i \in \mathcal{V}^a$ if and only if $\bigcap_i \mathcal{B}^{v_i}_{\mathcal{V}^{a,z_i}} \neq \phi$ (Cassandra, Littman, & Zhang 1997; Feng & Zilberstein 2004). Testing for this intersection requires solving a linear program that has $\sum_i |\mathcal{V}^{a,z_i}|$ constraints, one for each region constraint.

We note that the witness region of $v = \sum_i v_i \in \mathcal{V}^a$ is exactly the above intersection:

$$\mathcal{B}^v_{\mathcal{V}^a} = \bigcap_i \mathcal{B}^{v_i}_{\mathcal{V}^{a,z_i}}.$$

This gives us a way of relating the vectors in the output of the cross-sum stage, $\mathcal{V}^a$, to the regions defined by the vectors in the input vector sets $\{\mathcal{V}^{a,z_i}\}$. For each $v \in \mathcal{V}^a$, there is a corresponding list of vectors $\{v_1, v_2, \ldots, v_k\}$, where $v_i \in \mathcal{V}^{a,z_i}$, such that $v = \sum_i v_i$ and $\cap_i \mathcal{B}^{v_i}_{\mathcal{V}^{a,z_i}} \neq \phi$. We denote this list $parent(v)$.

Proposition 1 *The witness region of v is a subset of the witness region of any parent v_i:*

$$\mathcal{B}^v_{\mathcal{V}^a} \subseteq \mathcal{B}^{v_i}_{\mathcal{V}^{a,z_i}}; \tag{11}$$

Conversely, for each $v_i \in \mathcal{V}^{a,z_i}$, there is a corresponding lists of vectors $v^1, v^2, \ldots, v^m \in \mathcal{V}^a$, such that $v_i \in parent(v^j), \forall j$. We denote this list $child(v_i)$.

Proposition 2 *The witness region of v_i is the same as the union of its children's witness regions:*

$$\mathcal{B}^{v_i}_{\mathcal{V}^{a,z_i}} = \cup_j \mathcal{B}^{v^j}_{\mathcal{V}^a}. \tag{12}$$

The construction of the $parent$ and $child$ lists only requires some simple bookkeeping during the cross-sum stage. They will be the main building blocks of our algorithm.

Region-Based Maximization

Recall that in the maximization stage, the set $\mathcal{W} = \cup_a \mathcal{V}^a$ is pruned, where each $\mathcal{V}^a$ is obtained from the cross-sum pruning stage:

$$\mathcal{V}^a = \mathbb{PR}(\oplus_i \mathcal{V}^{a,z_i}).$$

Let us examine the process of pruning $\mathcal{W}$ using **procedure** $\mathbb{PR}$ in Table 1. In the `while` loop at line 14, an arbitrary vector $w \in \mathcal{W}$ is picked to compare with the current minimal set $\mathcal{D}$. As the size of $\mathcal{D}$ increases, the number of constraints in the linear programs approaches the size of the final result, $|\mathcal{V}'|$, leading to very large linear programs. However, to determine if some vector $w \in \mathcal{W}$ is dominated or not, we do not have to compare it with $\mathcal{D}$. Let $w \in \mathcal{V}^a$ and $v \in \mathcal{V}^{a'}$ for some a and a'.

Theorem 1 *If $a \neq a'$ and $\mathcal{B}^w_{\mathcal{V}^a} \cap \mathcal{B}^v_{\mathcal{V}^{a'}} = \phi$, then w is dominated by $\mathcal{W}$ if and only if w is dominated by $\mathcal{W} - v$.*

Proof: If w is dominated by $\mathcal{W}$, that is, $\forall b \in \mathcal{B}, \exists u \in \mathcal{W}$ such that $w \neq u$ and $w \cdot b < u \cdot b$. If $\mathcal{W} - v$ does not dominate w, then $\exists b' \in \mathcal{B}^v_{\mathcal{V}^{a'}}$ such that $\forall v' \in \mathcal{W} - v, w \cdot b' > v' \cdot b'$. Since $a \neq a', \forall v'' \neq w \in \mathcal{V}^a, w \cdot b' > v'' \cdot b'$ and therefore $b' \in \mathcal{B}^w_{\mathcal{V}^a}$. This contradicts the premise that $\mathcal{B}^w_{\mathcal{V}^a} \cap \mathcal{B}^v_{\mathcal{V}^{a'}} = \phi$. Therefore w must be dominated by $\mathcal{W} - v$.

If w is dominated by $\mathcal{W} - v$, then trivially it is also dominated by $\mathcal{W}$.■

Theorem 2 *If $a = a'$ and $\mathcal{B}^w_{\mathcal{V}^a} \cap \mathcal{B}^v_{\mathcal{V}^{a'} - w} = \phi$, then w is dominated by $\mathcal{W}$ if and only if w is dominated by $\mathcal{W} - v$.*

Proof: The proof is analogous to that of Theorem 1.■

Intuitively, the two theorems state that to test dominance for w, we only need to compare it with vectors that have a witness region overlapping with the witness region of w. (Although we frame the theorems for the case of maximization pruning, it can be easily generalized to the pruning of any set of vectors.) However, finding these overlapping vectors in general can be just as hard as the original pruning problem, if not harder. So this result does not translate to a useful algorithm in general. Fortunately, for maximization pruning, the special setting in which the union of some previously cross-summed vectors are pruned allows us to perform a close approximation of this idea efficiently. We present a simple algorithm for doing so next.

Algorithm

We start by finding vectors in $\mathcal{V}^a - w$ that have a witness region overlapping with the witness region of w. From Equation 11, each vector $v_i \in parent(w)$ has a witness region $\mathcal{B}^{v_i}_{\mathcal{V}^{a,z_i}}$ that fully covers the witness region of w. From Equation 12, each witness region $\mathcal{B}^{v_i}_{\mathcal{V}^{a,z_i}}$ is composed of witness regions of $child(v_i)$. Therefore the set

$$\mathcal{D}(w) = \{v | v \in child(v_i), v_i \in parent(w)\} \tag{13}$$

most likely contains vectors in $\mathcal{V}^a$ that have witness regions surrounding that of w, and their witness regions in the set $\mathcal{V}^a - w$ will overlap with the witness region of w.

Next we build a set of vectors in $\mathcal{V}^{a'}, a \neq a'$ that overlaps with the witness region of w. First, let $b(w)$ be the belief state that proved w is not dominated in $\mathcal{V}^a$. This belief state is obtained from solving the linear program during the cross-sum pruning stage. We can find in the vector set $\mathcal{V}^{a'}$ a vector $v_{a'}$ that has a witness region containing $b(w)$, using procedure BEST in Table 1:

$$v_{a'} = \text{BEST}(b(w), \mathcal{V}^{a'}).$$

By construction, $v_{a'}$ and w share at least a common belief state, $b(w)$. Now we use the same procedure as Equation 13 to build a set of vectors that covers the witness region of $v_{a'}$:

$$\mathcal{D}(v_{a'}) = \{v | v \in child(v_i), v_i \in parent(v_{a'})\}$$

Finally, we put together all these vectors:

$$\mathcal{D}' = \mathcal{D}(w) \cup \bigcup_{a' \neq a} \mathcal{D}(v_{a'}),$$

and use it to replace the set $\mathcal{D}$ at line 19 in Table 1 during maximization pruning. As a simple optimization, we replace $\mathcal{D}$ only when $|\mathcal{D}'| < |\mathcal{D}|$. The rest of the pruning algorithm remains the same.

Note that both $\mathcal{D}(w)$ and $\mathcal{D}(v_{a'})$ are incomplete. For $\mathcal{D}(w)$, it contains vectors that share a common parent with w, but there can be vectors that touch the boundary of the witness region of w but don't share the same parent with it. For $\mathcal{D}(v_{a'})$, besides the same problem, the witness region

				Time		#LP proj		Average #C proj		#LP max		Average #C max	
problem	$\|S\|$	$\|A\|$	$\|Z\|$	RBIP-M	RBIP	RBIP-M	RBIP	RBIP-M	RBIP	RBIP-M	RBIP	RBIP-M	RBIP
tiger	2	3	2	20.28	20.39	7292	5446	19.81	19.11	4535	4527	11.26	19.04
paint	4	4	2	27.55	27.72	5033	2736	14.15	13.86	3325	2820	6.40	15.96
shuttle	8	3	5	681.39	608.43	58937	58533	28.49	29.64	84086	86500	200.36	219.38
network	7	4	2	1367.68	1992.16	128132	118749	25.24	25.47	207909	204708	103.31	283.63
4x3	11	4	7	5529.90	41567.91	11622	10765	58.31	63.32	31828	36155	636.25	6646.32

Table 2: Comparisons between RBIP-M and RBIP. "#LP proj" is the number of linear programs solved during projection pruning. "Average #C proj" is the average number of constraints in the linear programs in the projection pruning. "#LP max" and "Average #C max" are the corresponding numbers for the maximization pruning stage. Time is in CPU seconds.

of $v_{a'}$ may only partially overlap with that of w. Therefore the set $\mathcal{D}'$ constructed above does not guarantee that a dominated vector can always be detected. This does not affect the correctness of the dynamic programming algorithm, however, because the resulting value function still accurately represents the true value, albeit with extra useless vectors. These useless vectors will be included as the input to the next DP update step, in which their projections (Equation 8) will be removed during the projection pruning stage (Equation 5). At the cross-sum stage (Equation 6), the input vectors become the same as those produced by a regular DP algorithm that does not use our maximization pruning technique. Therefore the extra computation caused by the inaccurate pruning of our algorithm in the previous DP step happens at the projection pruning stage only.

As we will see in the next section, this extra computation is usually insignificant compared to the savings obtained from the maximization step. This may seem counterintuitive because the pruning of those undetected dominated vectors is not avoided, but merely delayed to the next step of the DP update. However, as explain earlier, the projection usually maps into a small region of the belief space, resulting in a larger number of vectors being pruned from the projection. As a result, the linear programs in the projection pruning are usually much smaller than the ones in the previous maximization pruning stage.

It is possible that for some problems, the projection pruning is so efficient that the maximization pruning step can be skipped without significantly increasing the projection pruning time. However, even when this is true, pruning at the maximization step is still necessary, because it has impact on computing the error bound and detecting convergence, a process that involves solving linear programs similar to the ones used in the pruning process. Thus, skipping maximization pruning may greatly increase the time needed for computing the error bound. We leave the detailed analysis of this possibility to future work.

Experimental Results

Our algorithm is easy to implement and it only affects the maximization step of the standard DP update. There are many POMDP algorithms that use this standard DP update as a component. For example, Hansen's policy iteration algorithm uses standard DP update for policy improvement (Hansen 1998); Zhang & Zhang's point based value iteration interleaves standard DP update with point based DP update (Zhang & Zhang 2001); Feng & Hansen's approximate value iteration uses a symbolic representation of the value function in the standard DP update (Feng & Hansen 2001); Zhang & Zhang's restricted value iteration (Zhang & Zhang 2002) also uses the standard DP update with a transformed belief space. All these algorithms can be easily modified to incorporate the improvement offered by our technique. In this paper, we present experimental results on applying our algorithm to the RBIP algorithm as described in (Feng & Zilberstein 2004). We call our algorithm RBIP-M, and compare its performance against RBIP.

We test the algorithms on a set of benchmark problems from the literature. The number of states $|S|$, number of actions $|A|$ and number of observation states $|Z|$ of each problem are listed in Table 2. These problems are obtained from Cassandra's online repository (Cassandra 1999). All tests use a numerical precision of 10^{-6}. The algorithm is considered converged when the error bound is less than 0.01, except for problem `4x3` (see below). The machine used for testing is a dual-Athlon running at 1.2GHz. Only one CPU is used for the computation.

Our algorithm relies on two kinds of structures in a problem to perform well. First, the reachability and observability structure should be sparse so that the projection pruning can be much more efficient than the maximization pruning. The columns "Average #C proj" and "Average #C max" in Table 2 reflect this property. Second, the local structure of the belief regions defined by the vectors should allow neighboring relations among the regions to be adequately and efficiently captured by the *parent* and *child* lists. The adequacy is reflected by the "#LP proj" column, showing the extra number of linear programs that RBIP-M has to solve as a result of the undetected dominated vectors in the maximization stage. The efficiency is reflected by the reduction in the number of constraints in the maximization stage, shown in column "Average #C max".

For the problems `network` and `4x3`, RBIP-M is significantly faster than RBIP. (Coincidentally, these two problems are generally considered to be the harder problems in the literature.) This is because both structures are present in these problems. For example, in `4x3`, the average number of constraints in the projection pruning is about 60, much smaller than the number of constraints in the maximization stage. In addition, our algorithm is able to identify a much smaller set of vectors for use in the maximization linear programs (636.25 vs. 6646.32), while still effectively pruning most of the dominated vectors, resulting in only a small increase in the number of linear programs (from 10765 to 11622) solved during the projection stage. Combining these two factors gives our algorithm a great advantage. Note that for

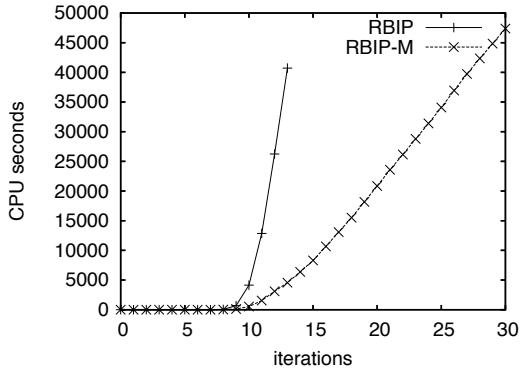

Figure 1: Running time comparison on problem 4x3.

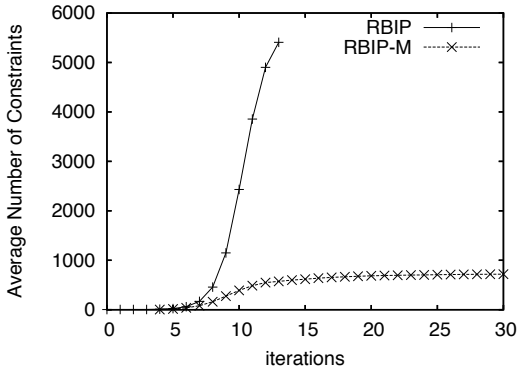

Figure 2: Average number of constraints in problem 4x3.

`4x3`, the data shown in Table 2 only represents the first 14 DP steps in both algorithms. At the end of the 14th iteration, RBIP already uses over 10 hours and is terminated. At this point RBIP-M is about 8 times faster than RBIP. The Bellman residual at this point is 0.06. We continue to run RBIP-M on the problem for another 16 steps, reducing the Bellman residual to 0.03 using about the same amount of time required for the 14 steps of RBIP. The running time of these steps are plotted in Figure 1, and the average number of constraints in the maximization pruning is plotted in Figure 2. From these figures, we infer that the actual speedup of RBIP-M over RBIP on this problem can be much greater.

For the other three problems, one of the two structures is absent, leading to little performance improvement. In `tiger` and `paint`, the first structure is missing, as reflected by the number of constraints during the projection pruning being comparable to that during the maximization pruning. As a result, even though the maximization pruning deals with much smaller linear programs, the saving is offset by the extra cost incurred during the subsequent projection pruning. In the problem `shuttle`, the second structure is missing, as reflected by the fact that the number of constraints in RBIP-M (200.36) is only slightly smaller than that in RBIP (219.38). Therefore there is not much saving gained in the maximization pruning step and RBIP-M runs slower than RBIP in this case due to the extra linear programs solved in the projection stage.

Conclusions

We have identified special properties of the projection and maximization stages in the DP process for POMDPs, and showed how they can be exploited to greatly accelerate the pruning process. Our technique is simple to implement and can be easily incorporated into several other POMDP algorithms. In future work, we will study how our approach improves other POMDP algorithms. This method promises to significantly alleviate the current computational bottlenecks in these algorithms.

Acknowledgment This work was supported in part by NSF grant number IIS-0219606.

References

Cassandra, A.; Littman, M.; and Zhang, N. 1997. Incremental pruning: A simple, fast, exact method for partially observable markov decision processes. In *Proc. of the 13th Conf. on Uncertainty in Artificial Intelligence*, 54–61.

Cassandra, A. R. 1999. Tony's POMDP page. http://www.cs.brown.edu/research/ai/pomdp/.

Feng, Z., and Hansen, E. 2001. Approximate planning for factored POMDPs. In *Proc. of the 6th European Conf. on Planning*.

Feng, Z., and Zilberstein, S. 2004. Region-based incremental pruning for pomdps. In *Proc. of the 20th Conf. on Uncertainty in Artificial Intelligence*, 146–153.

Hansen, E. A. 1998. An improved policy iteration algorithm for partially observable MDPs. In *Proc. of the 14th Conf. on Uncertainty in Artificial Intelligence (UAI-98)*.

Littman, M.; Cassandra, A.; and Kaelbling, L. 1996. Efficient dynamic-programming updates in partially observable markov decision processes. Technical Report CS-95-19, Brown University, Providence, RI.

Littman, M. 1994. The witness algorithm: Solving partially observable markov decision processes. Technical Report CS-94-40, Computer Science, Brown University.

Smallwood, R., and Sondik, E. 1973. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research* 21:1071–1088.

White, C. 1991. A survey of solution techniques for the partially observed markov decision process. *Annals of Operations Research* 32:215–230.

Zhang, N. L., and Liu, W. 1996. Planning in stochastic domains: Problem characteristics and approximation. Technical Report HKUST-CS96-31, Hong Kong University of Science and Technology.

Zhang, N., and Zhang, W. 2001. Speeding up the convergence of value iteration in partially observable markov decision processes. *Journal of AI Research* 14:29–51.

Zhang, W., and Zhang, N. 2002. Value iteration working with belief subset. In *Proc. of the 18th National Conf. on Artificial Intelligence (AAAI-02)*.