# Region-Based Dynamic Programming for Partially Observable Markov Decision Processes

Zhengzhu Feng \*

FENGZZ@CS.UMASS.EDU

Department of Computer Science University of Massachusetts Amherst, MA 01003

# Abstract

We present a major improvement to the dynamic programming (DP) algorithm for solving partially observable Markov decision processes (POMDPs). Our technique first targets the cross-sum pruning step of the DP update, a key source of complexity in POMDP algorithms. Unlike previous approaches, which reason about the whole belief space, the algorithms we present divide the belief space into smaller regions and perform independent pruning in each region. Because the number of useful vectors over each region can be much smaller than those over the whole belief space, we show that the linear programs used in the pruning process can be made exponentially smaller. With this exponential improvement to cross-sum pruning, we shift our attention to the next bottleneck, the maximization pruning step. Using the same region-based reasoning, we identify two types of structures in the belief space of a POMDP and show how to exploit them to reduce significantly the number of constraints in the linear programs used for maximization pruning. We discuss future research directions on extending these techniques to improve the scalability of POMDP algorithms.

# 1. 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 and have received a lot of attentions.

The first exact algorithm for solving general POMDPs was developed by Smallwood and Sondik (Sondik, 1971; Smallwood & Sondik, 1973; Sondik, 1978), and was subsequently improved by Cheng (Cheng, 1988). More recent work on the witness algorithm (Littman, 1994) and the incremental pruning algorithm (Zhang & Liu, 1996; Cassandra, Littman, & Zhang, 1997; Cassandra, 1998) provided a major speedup in exact algorithms. In particular, incremental pruning is currently considered the most efficient exact algorithm for performing the dynamic programming update for POMDPs. Most recent work on exact algorithms uses it as a basic building block. This paper further develops this line of work. There is also an extensive body of work on approximation methods. The earliest approach is probably the grid-based approach, which continues to be an active area of research (Drake,

<sup>\*.</sup> The author now works for Google Inc. This work was done while the author was at UMass.

1962; Lovejoy, 1991; Brafman, 1997; Hauskrecht, 1997; Zhou & Hansen, 2001). General function approximation methods also received significant attention (Hauskrecht, 2000). Chen (Cheng, 1988) presented some theoretical results on approximating the dynamic programming operators used by exact algorithms, and Feng and Hansen extended these results and applied them to a version of the incremental pruning algorithm that exploits state abstraction (Feng & Hansen, 2001, 2004). Other heuristic approaches include hierarchical decomposition (Theocharous & Mahadevan, 2002; Hansen & Zhou, 2003; Pineau, Gordon, & Thrun, 2003), direct policy search (Ng & Jordan, 2000; Poupart & Boutilier, 2003), and belief compression methods (Roy & Thrun, 1999; Roy & Gordon, 2003). Despite these extensive research efforts on both exact and approximate algorithms, the computational complexity of POMDPs still presents a major barrier that limits their applicability to small problems.

Many of the above algorithms rely on performing a dynamic programming (DP) update on the value function, represented by a finite set of linear vectors over the state space. A key source of complexity is the size of this representation, which in the worst case grows exponentially with the number of observations and the planning horizon. Fortunately, a large number of vectors in this representation can be pruned away without affecting the result. There is a standard linear programming (LP) method for detecting these useless vectors. Therefore, solving linear programs to prune useless vectors becomes the main computational component in the DP update.

Consequently, many research efforts have focused on improving the efficiency of pruning useless vectors. As will be described shortly in Section 3, 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 bottleneck in the DP update process. As a result, most research efforts focus on the cross-sum stage. The incremental pruning algorithm mentioned above, which is among the state-of-the-art techniques, is designed to address this problem. It interleaves the cross-sum and the pruning operators, which leads to significantly reduced number of linear programs to be solved in the cross-sum stage. In this paper, we introduce a major improvement to the incremental pruning technique. In particular, our work can be seen as a generalization of the *restricted region* variant of incremental pruning (Cassandra et al., 1997: Cassandra, 1998). The restricted region algorithm exploits the special structure in the cross-sum of two sets of vectors to reduce the number of constraints in the LPs. We show how this two-set restriction severely limits the applicability of incremental pruning, especially to problems with a large number of observations. We then show how to overcome this limitation so that the kind of structure exploited by RR can be extended to the whole cross-sum process. The resulting algorithm delivers an exponential speedup to the cross-sum stage.

With this, we shift our attention to the maximization stage. On the surface, the number of vectors only increases linearly during the maximization stage. However, the input to the maximization stage is the results of the cross-sum stage, which can be in itself of exponential size. Therefore the pruning in the maximization stage can often be as expensive as the crosssum stage. This is a less noticed bottleneck, since the cross-sum comes before maximization in the DP update process. In this paper, we identify two types of properties of the projection and maximization stages and show how they can be exploited to greatly accelerate the DP process. 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.

The algorithm presented in this paper preserves the simplicity of the original incremental pruning technique. Yet, it delivers superb performance improvements. It also preserves the generality of the original algorithm, and can thus be embedded into the more advanced algorithms as cited above that make use of the incremental pruning.

# 2. Partially Observable Markov Decision Processes

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. For simplicity we assume that all actions are applicable in all states;
- 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 actually sense;
- O is the observation model,  $O^a(z|s')$  is the probability that observation z is seen 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:

$$b: S \to [0, 1], \quad \sum_{s \in S} b(s) = 1.0.$$

Given a belief state b, representing the agent's current estimate of the underlying state, 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  $b' = T_z^a(b)$  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 state space that contains all possible belief states;
- A is the action set as in the POMDP model;
- T is the belief transition function as defined above; and
- $R_{\mathcal{B}}$  is the reward model, constructed from the POMDP model:

$$R^{a}_{\mathcal{B}}(b) = \sum_{s \in S} b(s) R^{a}(s).$$

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

$$V^{n}(b) = \max_{a \in A} \left\{ R^{a}_{\mathcal{B}}(b) + \beta \sum_{z \in Z} P^{a}(z|b) V^{n-1}(T^{a}_{z}(b)) \right\}.$$
 (1)

Given an arbitrary initial value function  $V^0$ , the sequence of value function produced by the DP update converges to the optimal value function  $V^*$ :

$$V^*(b) = \lim_{n \to \infty} V^n(b),$$

An optimal policy,  $\pi^* : \mathcal{B} \to A$ , which prescribes the optimal action to take in any given belief state, can be extracted from the optimal value function using a simple one-step lookahead:

$$\pi^*(b) = \arg\max_{a \in A} \left\{ R^a_{\mathcal{B}}(b) + \beta \sum_{z \in Z} P^a(z|b) V^*(T^a_z(b)) \right\}$$

An algorithm that computes the optimal value function directly by repeatedly applying Equation 1, and then extracting the control policy from the value function, is usually referred to as *value iteration* (Sondik, 1971). There is another type of algorithm, generally referred to as *policy iteration*, that computes the policy directly (Sondik, 1971; Hansen, 1998). It represents a policy explicitly and interleaves two steps to improve the policy: the *policy evaluation* step computes a value function that represents the value of executing the current policy, and the *policy improvement* step uses Equation 1 to update the value function and extract from it an improved policy. For both types of POMDP algorithms, the DP update is a central computational component which is why it has been the focus of many research efforts.

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, \dots, 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. Among several algorithms that have been developed to perform this DP step, incremental pruning (IP) is considered the most efficient.

# 3. Incremental Pruning

Note that the DP update in Equation 1 can be expressed as a combination of simpler functions (Cassandra et al., 1997):

$$V^{a,z}(b) = \frac{R^a_{\mathcal{B}}(b)}{|Z|} + \beta P^a(z|b)V^{n-1}(T^a_z(b))$$
$$V^a(b) = \sum_{z \in Z} V^{a,z}(b)$$
$$V^n(b) = \max_{a \in A} V^a(b)$$

Each of these functions is piecewise linear and convex, and can be represented by a unique minimum-size set of vectors. We use the symbols  $\mathcal{V}^n$ ,  $\mathcal{V}^a$ , and  $\mathcal{V}^{a,z}$  to refer to these minimum-size sets, and use  $\mathcal{V}^{n-1}$  to refer to the set of vectors representing the previous value function  $V^{n-1}$ .

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 as:

$$\mathcal{U} \oplus \mathcal{W} = \{ u + w | u \in \mathcal{U}, w \in \mathcal{W} \}.$$
 (2)

Note that

$$|\mathcal{U}\oplus\mathcal{W}|=|\mathcal{U}|\times|\mathcal{W}|.$$

An operator that takes a set of vectors  $\mathcal{W}$  and reduces it to its unique minimum form is denoted  $\mathbb{PR}(\mathcal{W})$ . We also use  $\mathbb{PR}(\mathcal{W})$  to denote the resulting minimum set. Formally,

$$w \in \mathbb{PR}(\mathcal{W}) \iff w \in \mathcal{W}, \text{and } \exists b \in \mathcal{B} \text{ such that } \forall w' \neq w \in \mathcal{U}, w \cdot b > w' \cdot b.$$

Using this notation, the minimum-size sets of vectors defined earlier can be computed as follows:

$$\mathcal{V}^{a,z} = \mathbb{P}\mathbb{R}\left(\{v^{a,z,i} | v^i \in \mathcal{V}^{n-1}\}\right),\tag{3}$$

$$\mathcal{V}^a = \mathbb{P}\mathbb{R}\left(\oplus_{z \in Z} \mathcal{V}^{a, z}\right) \tag{4}$$

$$\mathcal{V}^n = \mathbb{PR}\left(\cup_{a \in A} \mathcal{V}^a\right) \tag{5}$$

where  $v^{a,z,i}$  is the vector computed by

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

The three steps are usually referred to as the projection stage (3), cross-sum stage (4) and maximization stage (5).

Table 1 summarizes an algorithm, described in (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 tests for dominated vectors. The simpler 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 cannot detect all dominated vectors.

There is a linear programming method that can detect all dominated vectors. Given a vector w and a set of vectors  $\mathcal{D}$  that does not include w, the linear program in procedure LP-DOMINATE of Table 1 determines whether adding w to  $\mathcal{D}$  improves the value function represented by  $\mathcal{D}$  for any belief state b. If it does, the variable d optimized by the linear program is the maximum amount by which the value function is improved, and b is the belief state that optimizes d. If it does not, that is, if  $d \leq 0$ , then w is dominated by  $\mathcal{D}$ .

The algorithm summarized in Table 1 uses these two tests for dominated vectors to prune a set of vectors to its minimum size. The symbol  $<_{lex}$  in the pseudo-code denotes lexicographic ordering. Littman (1994) elucidated its significance in implementing this algorithm.

Since the linear programming method takes up most of the computation time in the pruning, to simplify our discussion, we will omit analyzing the point-wise domination test in the rest of the paper. We can assume either that the point-wise domination test is always performed before pruning since it takes little computation time, or that we don't use the point-wise domination test at all since it can only detect a small number of dominated vectors.

As we can see from the table, to prune a vector set  $\mathcal{W}$ , we need to solve a linear program for each vector in  $\mathcal{W}$ . In other words, to prune the set  $\mathcal{W}$  we need to solve  $|\mathcal{W}|$  LPs. Among the three pruning steps, Equations (5) and (3) can be carried out relatively efficiently with respect to their input size. Equation (4) presents a major bottleneck because the size of the cross-sum is the product of the inputs:  $|\mathcal{U} \oplus \mathcal{W}| = |\mathcal{U}| \times |\mathcal{W}|$ . As a result, it is necessary to process  $\prod_{z} |\mathcal{V}^{a,z}|$  vectors in computing  $\mathcal{V}^{a}$ . This translates into solving  $\prod_{z} |\mathcal{V}^{a,z}|$  LPs. Incremental pruning (IP) is designed to specifically addresses this problem. It exploits the fact that the  $\mathbb{PR}$  and  $\oplus$  operators can be interleaved:

$$\mathbb{PR}(\mathcal{U} \oplus \mathcal{V} \oplus \mathcal{W}) = \mathbb{PR}(\mathcal{U} \oplus \mathbb{PR}(\mathcal{V} \oplus \mathcal{W})).$$
(6)

procedure POINTWISE-DOMINATE $(w, \mathcal{D})$ 1. for each  $u \in \mathcal{D}$ 2. if  $w(s) \leq u(s), \forall s \in S$  then return true 3. return false procedure LP-DOMINATE $(w, \mathcal{D})$ 4. solve the following linear program variables:  $d, b(s) \forall s \in S$ maximize dsubject to the constraints  $b \cdot (w - u) \ge d, \ \forall u \in \mathcal{D}$  $\sum_{s \in S} b(s) = 1$ 5. if  $d \ge 0$  then return b 6. else return nil procedure BEST(b, W)7.  $max \leftarrow -\infty$ 8. for each  $u \in \mathcal{W}$ 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 wprocedure  $\mathbb{PR}(\mathcal{W})$ 13.  $\mathcal{D} \leftarrow \emptyset$ 14. while  $\mathcal{W} \neq \emptyset$ 15. $w \leftarrow \text{any element in } \mathcal{W}$ 16.if POINTWISE-DOMINATE $(w, \mathcal{D})$  = true 17. $\mathcal{W} \leftarrow \mathcal{W} - \{w\}$ 18. else  $b \leftarrow \text{LP-DOMINATE}(w, \mathcal{D})$ 19. 20.if b = nil then 21. $\mathcal{W} \leftarrow \mathcal{W} - \{w\}$ 22.else 23. $w \leftarrow \text{BEST}(b, \mathcal{W})$  $\mathcal{D} \leftarrow \mathcal{D} \cup \{w\}$ 24. $\mathcal{W} \leftarrow \mathcal{W} - \{w\}$ 25.26. return  $\mathcal{D}$ 

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

Thus Equation (4) can be computed as follows:

$$\mathcal{V}^{a} = \mathbb{PR}(\mathcal{V}^{a, z_{1}} \oplus \mathbb{PR}(\mathcal{V}^{a, z_{2}} \oplus \cdots \mathbb{PR}(\mathcal{V}^{a, z_{k-1}} \oplus \mathcal{V}^{a, z_{k}}) \cdots)), \tag{7}$$

which is what the IP algorithm does. The benefit of IP is the reduction of the number of LPs that need to be solved. This can best be understood when Equation (7) is viewed as a recursive process: Instead of pruning the cross-sum  $\bigoplus_{z \in \mathbb{Z}} \mathcal{V}^{a,z}$  directly, IP breaks it down

by recursively computing  $\mathbb{PR}(\bigoplus_{i=2}^{k} \mathcal{V}^{a,z_i})$  first, and then prune the cross-sum

$$\mathcal{V}^{a,z_1} \oplus \mathbb{PR}(\oplus_{i=2}^k \mathcal{V}^{a,z_i}).$$

Because the size of  $\mathbb{PR}(\bigoplus_{i=2}^{k} \mathcal{V}^{a,z_i})$  is potentially much smaller than  $\prod_{i=2}^{k} |\mathcal{V}^{a,z_i}|$ , the number of LPs needed to prune  $\bigoplus_{z \in \mathbb{Z}} \mathcal{V}^{a,z}$  is reduced from  $\prod_z |\mathcal{V}^{a,z}|$  to

$$|\mathcal{V}^{a,z_1}| \times |\mathbb{PR}(\oplus_{i=2}^k \mathcal{V}^{a,z_k})|$$

Note that this argument applies equally to the recursive step  $\mathbb{PR}(\bigoplus_{i=2}^{k} \mathcal{V}^{a,z_i})$ . In general, the total number of LPs used by IP and its variants in computing Equation (4) is asymptotically  $|\mathcal{V}^a|\sum_z |\mathcal{V}^{a,z}|$  (Cassandra, 1998). Note that for typical POMDPs,  $|\mathcal{V}^a|$  is usually smaller than, but nevertheless on the same order as,  $\prod_z |\mathcal{V}_z^a|$ .

Another bottleneck in computing Equation (4) is caused by the number of constraints in each of the linear programs that needs to be solved. From the procedure LP-DOMINATE in Table 1, each linear program solved has  $|\mathcal{D}|$  inequality constraints, where  $|\mathcal{D}|$  eventually approaches  $|\mathcal{V}^a|$  when computing Equation (4). Again, this is exponential in the size of the previous value function. Although IP can effectively reduce the number of linear programs that need to be solved, it does not address the issue of the number of constraints. As a result, when using IP to solve POMDPs, especially those with a large number of observations, the large number of linear programs is usually not the first obstacle that we encounter. Instead, what we usually observe is that the program gets stuck solving one of the linear programs, because it has too large a number of constraints. Our main contribution is to show how the number of constraints can be reduced dramatically without affecting the solution quality, while at the same time maintaining the same number of linear programs as IP.

#### 4. Witness region

Recall that the value function of a POMDP is piece-wise linear and convex (PWLC), and there is a unique and minimal vector representation for a PWLC function. Figure 1 shows an example of a value function minimally represented by three vectors. In such a representation, each vector  $u \in \mathcal{U}$  defines a *witness region*  $\mathcal{B}^u_{\mathcal{U}}$  over which u dominates all other vectors in  $\mathcal{U}$  (Littman, Cassandra, & Kaelbling, 1996):

$$\mathcal{B}^{u}_{\mathcal{U}} = \{b|b \cdot (u - u') > 0, \forall u' \in \mathcal{U} - \{u\}\}.$$
(8)

For simplicity of notation, we use  $\mathcal{B}_{\mathcal{U}}$  to refer to a belief region defined by some vector in  $\mathcal{U}$ , when the specific vector is irrelevant or understood from the context. We also use  $\tilde{\mathcal{B}}$  to refer to some region when the vector and vector set are irrelevant or understood from the context.

Note that each inequality in Equation (8) 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  $\mathbb{L}(\mathcal{B}^u_{\mathcal{U}}) := \{(u - u') | u' \in \mathcal{U} - \{u\}\}$  to represent the set of region constraints defining  $\mathcal{B}^u_{\mathcal{U}}$ . Note that for any two regions  $\mathcal{B}^u_{\mathcal{U}}$  and  $\mathcal{B}^w_{\mathcal{W}}$ ,

$$\mathbb{L}(\mathcal{B}^{u}_{\mathcal{U}} \cap \mathcal{B}^{w}_{\mathcal{W}}) = \mathbb{L}(\mathcal{B}^{u}_{\mathcal{U}}) \cup \mathbb{L}(\mathcal{B}^{w}_{\mathcal{W}}).$$
(9)



Figure 1: Witness region

For each value function, there is an associated set of witness-regions that represents a partition of the belief-state space. Throughout the dynamic programming process, the value function is always finite, giving us a finite number of regions as well. We call this representation of the belief-state space a region-based representation. Note that the region-based representation is always associated with a vector representation of the value function and no new data structure is required to represent it. Therefore the region-based representation is more of a change of perspective when looking at the value functions of a POMDP. As I will show in this chapter, this change in perspective brings a dramatic improvement to the algorithm.

# 5. Region-based cross-sum pruning

In this section, we show how the explicit region-based belief-state representation can be exploited to greatly increase the performance of the cross-sum pruning operation. To simplify the notation, we drop the a and z superscripts in the cross-sum pruning Equation (4), and refer to the computation as

$$\mathcal{V} = \mathbb{PR}(\oplus_{i=1}^{k} \mathcal{V}_i). \tag{10}$$

Furthermore, we omit specifying the range of i when the range is from 1 to k.

Consider the cross-sum set  $\mathcal{U} \oplus \mathcal{W}$ , where  $\mathcal{U}$  and  $\mathcal{W}$  are assumed to be minimal. It has been observed that (Cassandra, 1998):

**Theorem 1** Let  $u \in \mathcal{U}$  and  $w \in \mathcal{W}$ . Then  $(u+w) \in \mathbb{PR}(\mathcal{U} \oplus \mathcal{W})$  if and only if  $\mathcal{B}^u_{\mathcal{U}} \cap \mathcal{B}^w_{\mathcal{W}} \neq \phi$ .

**Proof** If  $(u+w) \in \mathbb{PR}(\mathcal{U} \oplus \mathcal{W})$ , then  $\exists b \in \mathcal{B}$  such that  $\forall (u'+w') \in \mathcal{U} \oplus \mathcal{W}$ ,

if 
$$(u+w) \neq (u'+w')$$
, then  $(u+w) \cdot b > (u'+w') \cdot b$ .

It follows that

$$\forall u' \neq u \in \mathcal{U}, \ (u+w) \cdot b > (u'+w) \cdot b,$$

**procedure** LP-INTERSECT( $\mathcal{B}_{\mathcal{V}_1}^{v_1}, \mathcal{B}_{\mathcal{V}_2}^{v_2}, \ldots, \mathcal{B}_{\mathcal{V}_k}^{v_k}$ )

1. construct the following linear program: variables:  $b(s) \forall s \in S$ maximize 0 subject to the constraints  $b \cdot (v_1 - v) > 0, \ \forall v \in \mathcal{V}_1 - \{v_1\}$  $b \cdot (v_2 - v) > 0, \ \forall v \in \mathcal{V}_2 - \{v_2\}$  $\vdots$  $b \cdot (v_k - v) > 0, \ \forall v \in \mathcal{V}_k - \{v_k\}$  $\sum_{s \in S} b(s) = 1$ 2. if the linear program is feasible, return **TRUE** 3. else return **FALSE** 

Table 2: Linear programming test for region intersection.

therefore  $u \cdot b > u' \cdot b$  and  $b \in \mathcal{B}^u_{\mathcal{U}}$ . Similarly,  $b \in \mathcal{B}^w_{\mathcal{W}}$ . Thus  $b \in \mathcal{B}^u_{\mathcal{U}} \cap \mathcal{B}^w_{\mathcal{W}}$  which implies  $\mathcal{B}^u_{\mathcal{U}} \cap \mathcal{B}^w_{\mathcal{W}} \neq \phi$ .

If  $\mathcal{B}^u_{\mathcal{U}} \cap \mathcal{B}^w_{\mathcal{W}} \neq \phi$ , then  $\exists b \in \mathcal{B}^u_{\mathcal{U}} \cap \mathcal{B}^w_{\mathcal{W}}$ , and so  $b \in \mathcal{B}^u_{\mathcal{U}}$  and  $b \in \mathcal{B}^w_{\mathcal{W}}$ . Thus

$$\forall u' \neq u \in \mathcal{U}, \ u \cdot b > u' \cdot b,$$

and

$$\forall w' \neq w \in \mathcal{W}, \ u \cdot b > u' \cdot b.$$

It follows that

$$\forall (u''+w'') \neq (u+w) \in \mathcal{U} \oplus \mathcal{W}, \ (u+w) \cdot b > (u''+w'') \cdot b.$$

Thus  $(u+w) \in \mathbb{PR}(\mathcal{U} \oplus \mathcal{W})$ .

This conclusion can be easily generalized to the cross-sum of more than two sets:

**Corollary 1** Let  $\mathcal{V}_i, i \in [1, k]$  be sets of vectors. Let  $v_i \in \mathcal{V}_i$ . Then  $\sum_{i=1}^k v_i \in \mathbb{PR}(\bigoplus_{i=1}^k \mathcal{V}_k)$  if and only if  $\bigcap_{i=1}^k \mathcal{B}^{v_i} \neq \phi$ .

With the region-based representation for the belief state space, testing for region intersection can easily be accomplished by solve a linear program to test if the individual regions share a common belief point. The linear program is listed in the procedure LP-INTERSECT in Table 2. We call this linear program the intersection LP. The constraints of the intersection LP is simply the combination of the region constraints of each region, plus the simplex constraint of the belief state b. In other words, the size of the intersection LP is  $\sum_{i=1}^{k} |\mathcal{V}_i| + 1$ .

# 5.1 Intersection-based incremental pruning

Corollary 1 suggests that the problem of computing  $\mathbb{PR}(\oplus_i \mathcal{V}_i)$  is equivalent to finding all intersecting regions defined by the different vector sets. We introduce the operator  $\mathbb{I}(\{\mathcal{V}_i\})$ 

that takes as input a set of vector sets and produces a list of intersecting regions defined by those vector sets:

$$\mathbb{I}(\mathcal{V}_{i_1},\ldots,\mathcal{V}_{i_t}) = \left\{ (\mathcal{B}_{\mathcal{V}_{i_1}}^{v_1},\ldots,\mathcal{B}_{\mathcal{V}_{i_t}}^{v_t}) | \cap_{j=1}^t \mathcal{B}_{\mathcal{V}_{i_j}}^{v_j} \neq \phi \right\}$$

Pruning of the cross-sums can then be expressed as

$$\mathbb{PR}(\oplus_i \mathcal{V}_i) = \left\{ \sum_i v_i \middle| (\mathcal{B}_{\mathcal{V}_1}^{v_1}, ..., \mathcal{B}_{\mathcal{V}_k}^{v_k}) \in \mathbb{I}(\mathcal{V}_1, ..., \mathcal{V}_k) \right\}$$
(11)

A naive approach to compute  $\mathbb{I}(\{\mathcal{V}_i\})$  is to enumerate all possible combinations of  $\{\mathcal{B}_{\mathcal{V}_i}\}$ , and test them for intersection using the intersection LP. This requires a total of  $\prod_i |\mathcal{V}_i|$  LPs, but each LP has only  $\sum_i |\mathcal{V}_i|$  constraints. A better approach would be to use an incremental process similar to IP: To compute  $\mathbb{I}(\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_k)$ , we test if

LP-INTERSECT(
$$\mathcal{B}_{\mathcal{V}_1}, \mathcal{B}_{\mathcal{V}_2}, \ldots, \mathcal{B}_{\mathcal{V}_k}$$
)

is true for all combinations of  $\mathcal{B}_{\mathcal{V}_1}$  and  $(\mathcal{B}_{\mathcal{V}_2},\ldots,\mathcal{B}_{\mathcal{V}_k})$ , where

$$(\mathcal{B}_{\mathcal{V}_2},\ldots,\mathcal{B}_{\mathcal{V}_k})\in\mathbb{I}(\mathcal{V}_2,\ldots,\mathcal{V}_k),$$

and  $\mathbb{I}(\mathcal{V}_2, \ldots, \mathcal{V}_k)$  is computed recursively in the same manner. The recursion stops at  $\mathbb{I}(\mathcal{V}_{k-1}, \mathcal{V}_k)$ , at which point the naive approach is used to compute the results. We call this algorithm for computing  $\mathbb{I}$  and subsequently  $\mathbb{PR}(\oplus_i \mathcal{V}_i)$  the *intersection-based incremental pruning* (IBIP).

Surprisingly, IBIP solves the exact same number of LPs as IP (and the RR variants). To see this, consider the top level of the recursion. The total number of combinations between  $\mathcal{B}_{\mathcal{V}_1}$  and  $(\mathcal{B}_{\mathcal{V}_2},\ldots,\mathcal{B}_{\mathcal{V}_k})$ , and hence the number of LPs needed, is

$$|\mathcal{V}_1| \times |\mathbb{I}(\mathcal{V}_2, \dots, \mathcal{V}_k)| = |\mathcal{V}_1| \times |\mathbb{P}\mathbb{R}(\bigoplus_{i=2}^k \mathcal{V}_i)|,$$

which is also the number of LPs needed at the top recursion of IP (see end of Section 3). Similarly the same numbers of LPs are solved at all recursive steps. It follows that the total numbers of LPs of the two approaches are the same:  $|\mathcal{V}| \sum |\mathcal{V}_i|$ .

However, all the LPs used in computing I have at most  $\sum_{i=1}^{k} |\mathcal{V}_i|$  constraints. In particular, when computing  $\mathbb{I}(\mathcal{V}_t, \ldots, \mathcal{V}_k)$ , the number of constraints ranges between  $\sum_{i=t+1}^{k} |\mathcal{V}_i|$  and  $\sum_{i=t}^{k} |\mathcal{V}_i|$ . Thus, to compute  $\mathbb{PR}(\bigoplus_i \mathcal{V}_i)$ , the IBIP algorithm requires the same number of LPs but with possibly an exponential reduction in the number of constraints compared to IP. The number of constraints does not depend on the size of the output set, as with IP and RR.

#### 5.2 Region-based incremental pruning

In this section, we show how the number of constraints in IBIP can be further reduced. To make a direct comparison with the recursion in IBIP, we will start from  $\mathcal{V}_k$ : To compute  $\mathbb{I}(\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_k)$ , we first fix a region in  $\mathcal{V}_k$ , call it  $\mathcal{B}_{\mathcal{V}_k}$ , and find all the elements in  $\mathbb{I}(\mathcal{V}_1, \ldots, \mathcal{V}_{k-1})$  that intersect with  $\mathcal{B}_{\mathcal{V}_k}$ . We repeat this for all the regions in  $\mathcal{V}_k$ . procedure  $\mathbb{I}^*(\tilde{\mathcal{B}}, \{\mathcal{V}_i | i \in [1, t]\})$ 1.  $\mathcal{K} \leftarrow \phi$ 2. if t = 1 $\mathcal{K} \leftarrow \{\mathcal{B}_{\mathcal{V}_1}^v | v \in \mathbb{PR}(\tilde{\mathcal{B}}, \mathcal{V}_1)\}$ 3. 4. else for each  $v \in \mathcal{V}_t$ 5. $\begin{aligned} \mathcal{V}'_i \leftarrow \mathbb{PR}(\tilde{\mathcal{B}} \cap \mathcal{B}^v_{\mathcal{V}_t}, \mathcal{V}_i), i \in [1, t-1] \\ \text{if } \exists i \in [1, t-1] \text{ such that } \mathcal{V}'_i = \phi \end{aligned}$ 6. 7. 8. continue 9.  $\mathcal{D} \leftarrow \mathbb{I}^*(\tilde{\mathcal{B}} \cap \mathcal{B}^v_{\mathcal{V}_t}, \{\mathcal{V}'_i | i \in [1, t-1]\})$  $\mathcal{K} \leftarrow \mathcal{K} \cup \{(\mathcal{B}_{\mathcal{V}_1}, ..., \mathcal{B}_{\mathcal{V}_{t-1}}, \mathcal{B}_{\mathcal{V}_t}^v) | (\mathcal{B}_{\mathcal{V}_1}, ..., \mathcal{B}_{\mathcal{V}_{t-1}}) \in \mathcal{D}\}$ 10. 11. return  $\mathcal{K}$ procedure  $\mathbb{I}(\mathcal{V}_1, \mathcal{V}_2, \dots, \mathcal{V}_k)$ 12. return  $\mathbb{I}^*(\mathcal{B}, \{\mathcal{V}_i | i \in [1, k]\})$ 

Table 3: Region-based pruning for computing  $\mathbb{I}$ .

To find all the regions in  $\mathbb{I}(\mathcal{V}_1, \ldots, \mathcal{V}_{k-1})$  that intersect with  $\mathcal{B}_{\mathcal{V}_k}$ , we first find all regions in each  $\mathcal{V}_i(1 \leq i \leq k-1)$  that intersect with  $\mathcal{B}_{\mathcal{V}_k}$ . Recall that each such region corresponds to a vector in the vector set, and the set of intersecting regions corresponds to some subset of vectors  $\mathcal{V}'_i \subseteq \mathcal{V}_i$ .  $\mathcal{V}'_i$  can be precisely computed by the region-based pruning,  $\mathcal{V}'_i =$  $\mathbb{PR}(\mathcal{B}_{\mathcal{V}_k}, \mathcal{V}_i)$ . Once all  $\mathcal{V}'_i$  are computed, we then recursively compute  $\mathbb{I}(\mathcal{V}'_1, \ldots, \mathcal{V}'_{k-1})$ , by fixing a  $\mathcal{B}_{\mathcal{V}'_{k-1}}$  and then find all the elements in  $\mathbb{I}(\mathcal{V}'_1, \ldots, \mathcal{V}'_{k-2})$  that intersect  $\mathcal{B}_{\mathcal{V}_k} \cap \mathcal{B}_{\mathcal{V}'_{k-1}}$ . Note that the  $\mathbb{I}$  operator serves only as a conceptual place-holder in this process; all the computations are carried out using the region-based pruning operator.

Table 3 shows the algorithm that finds the set of intersecting regions using this process. We call the algorithm that computes  $\mathbb{PR}(\oplus_i \mathcal{V}_i)$  using Table 3 and Equation (11) the regionbased incremental pruning (RBIP) algorithm.

The main motivation for RBIP is to further reduce the number of constraints. As Table 3 shows, all pruning in RBIP is of the form  $\mathbb{PR}(\tilde{\mathcal{B}}, \mathcal{V}_t)$ . In line 3, the pruning corresponds to testing some  $\mathcal{B}_{\mathcal{V}_1}$  with some  $(\mathcal{B}_{\mathcal{V}_2}, \ldots, \mathcal{B}_{\mathcal{V}_k})$  for intersection in IBIP. The number of constraints in IBIP is from  $\sum_{i=2}^{k} |\mathcal{V}_i|$  to  $\sum_{i=1}^{k} |\mathcal{V}_i|$ . The number of constraints in RBIP ranges between  $\sum_{i=2}^{k} |\mathcal{V}_i^*|$  and  $\sum_{i=1}^{k} |\mathcal{V}_i^*|$ , where  $\mathcal{V}_i^*$  is  $\mathcal{V}_i$  pruned multiple times previously in line 6. Because of the region-based pruning,  $|\mathcal{V}_i^*|$  could be much smaller than  $|\mathcal{V}_i|$  and this is where the savings come from. The analysis of the pruning in line 6 follows similarly.

In addition to reducing the number of constraints, RBIP can also reduce the number of linear programs. In Table 3, during each recursive call to  $\mathbb{I}^*()$ , the sizes of input sets  $\mathcal{V}'_i$  are already reduced by pruning. Thus each subsequent problem that  $\mathbb{I}^*()$  solves can be progressively smaller. However, without assuming any special restriction on the geometric form of the value function, it is also possible that the region-based pruning in line 6 may not prune any vector at all. In this case there is no saving in the number of constraints as compared to IBIP. Further, if every region-based pruning falls in this worst-case scenario, the total number of LPs solved by RBIP will be  $|Z||\mathcal{V}| \sum |\mathcal{V}_i|$ , or |Z| times that of IBIP. It remains an open question whether this happens in realistic POMDPs. In all the experiments we have performed so far, we observed substantial savings in terms of both the number of LPs and the number of constraints using RBIP (Feng & Zilberstein, 2004).

#### 6. Region-based 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  $\mathcal{V}$ . There has been relatively little work addressing maximization pruning, partly because the cross-sum pruning was the major bottle-neck that come before the maximization pruning step. With an exponential speedup to the cross-sum pruning, we are at a good position to address the maximization pruning step.

This section presents a simple algorithm for selecting constraints for the linear programs used in the maximization pruning stage. It exploits the locality structure of the belief state space. We show how the region-based representation makes it possible to reason explicitly about these structures.

#### 6.1 Projection pruning

Given the input value function  $\mathcal{V}$ , the linear programs in the projection pruning (Equation 3) 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 the back projection in Equation 3. The number of vectors needed to represent  $\mathcal{V}$  over this 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 property in our algorithm, by shifting some of the pruning in the maximization stage to the projection stage of the next DP update.

#### 6.2 Locality in belief space

Let

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

refer to a vector in the cross-sum, implying  $v_i \in \mathcal{V}^{a,z_i}$ . From Corollary 1,  $\sum_i v_i \in \mathcal{V}^a$  if and only if  $\bigcap_i \mathcal{B}_{\mathcal{V}^{a,z_i}}^{v_i} \neq \phi$ . Note that the witness region of  $v = \sum_i v_i \in \mathcal{V}^a$  is exactly this 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  $\bigcap_i \mathcal{B}_{\mathcal{V}^a,z_i}^{v_i} \neq \phi$ . We denote this list *parent*(v).

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

$$\mathcal{B}^{v}_{\mathcal{V}^{a}} \subseteq \mathcal{B}^{v_{i}}_{\mathcal{V}^{a,z_{i}}}; \tag{12}$$

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:

$$\mathcal{B}^{v_i}_{\mathcal{V}^a, z_i} = \bigcup_j \mathcal{B}^{v^j}_{\mathcal{V}^a}.\tag{13}$$

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.

# 6.3 Region-based maximization

Recall that in the maximization stage, the set  $\mathcal{W} = \bigcup_a \mathcal{V}^a$  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 (Page 7). 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 2** If  $a \neq a'$  and  $\mathcal{B}_{\mathcal{V}^a}^w \cap \mathcal{B}_{\mathcal{V}^{a'}}^v = \phi$ , 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  $\mathcal{W}$ .

**Corollary 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  $\mathcal{W} - v$ .

Intuitively, 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 theorem 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.

# 6.4 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 12, each vector  $v_i \in parent(w)$  has a witness region  $\mathcal{B}_{\mathcal{V}^a,z_i}^{v_i}$  that fully covers the witness region of w. From Equation 13, each witness region  $\mathcal{B}_{\mathcal{V}^a,z_i}^{v_i}$  is composed of witness regions of  $child(v_i)$ . Therefore the set

$$\mathcal{D}(w) = \{v | v \in child(v_i), v_i \in parent(w)\}$$
(14)

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 14 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 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 be always 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 will be removed during the projection pruning stage (Equation 3). At the cross-sum stage (Equation 4), 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.

# 7. Discussions

#### 7.1 Cross-sum pruning

For incremental pruning and its variants, the main computational bottleneck is the crosssum step. The main cause for this bottleneck is usually not the large number of linear programs. Rather, one prohibitively large linear program is enough to block the whole algorithm. We have successfully reduced the size of the linear programs involved in this process from having an exponential number of constraints to having a linear number of constraints. This essentially eliminated one source of complexity in POMDP algorithms.

However, the exponential improvement to the cross-sum step does not necessarily translate to exponential improvement to the whole DP algorithm. For hard POMDP problems, the number of minimum vectors necessary to represent the value function increases exponentially during each cross-sum. With RBIP or IBIP, we are not avoiding this exponential increase, we merely generate the (exponentially many) vectors exponentially faster in the cross-sum step.

We envision two research directions to further improve the performance of the cross-sum pruning step. Firstly, in our previous work, we showed that by pruning vectors that are not dominated but only contribute marginally to the value function, we can significantly reduce the number of vectors, while still maintain a reasonable approximation bound (Feng & Hansen, 2001). This technique can be applied to RBIP and IBIP. Secondly, the regionbased formulation of the cross-sum pruning makes it possible to carry out the computation in parallel with little overhead. For IBIP (Section 5.1), the intersection test for each individual combination of vectors can be carried out independently in parallel. For RBIP, the **for** loop in line 7 of the algorithm (Table 3) can be unrolled into independent processes and carried out in parallel. With the right infrastructure, we can significantly scale up the cross-sum algorithm.

# 7.2 Maximization pruning

We have shown how to exploit the local structure of the belief space during the maximization pruning step of the DP algorithm. As we demonstrated in (Feng & Zilberstein, 2005), the effectiveness of our algorithm relies on two kinds of structures, namely locality and reachability. With locality, we only need to look at vectors whose witness regions are close to that of the vector we are testing for dominance. With reachability, we can safely "leak" vectors to the next projection step without incurring too much penalty. Ideally we would want to prevent the leaking altogether so that the algorithm can be less dependent on this structure. One way is to select the vectors more intelligently. Currently when selecting the vector  $v_{a'}$  from  $\mathcal{V}^{a'}$ , (Section 6.4), we only looked at one witness point. It is possible to select the vectors more intelligently and reduce the leaking. A better approach may be to keep track of the neighboring relations of the witness regions across multiple DP steps, which would eliminate the reliance on the reachability structure.

#### References

- Andersen, E. D., & Andersen, K. D. (1995). Presolving in linear programming. Mathematical Programming, 71(2), 221–245.
- Brafman, R. (1997). A heuristic variable grid solution method for POMDPs. In Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97), Providence, RI.
- Brearly, A. L., Mitra, G., & Williams, H. P. (1975). An analysis of mathematical programming problems prior to applying the simplex method. *Mathematical Programming*, 8, 54–83.
- Cassandra, A., Littman, M. L., & Zhang, N. L. (1997). Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In *Proceedings of* the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence, pp. 54–61.
- Cassandra, A. R. (1998). Exact and Approximate Algorithms for Partially Observable Markov Decision Processes. Ph.D. thesis, Brown University.
- Cheng, H. (1988). Algorithms for Partially Observable Markov Decision Processes. Ph.D. thesis, University of British Columbia.
- Drake, A. (1962). Observation of Markov Process Through a Noisy Channel. Ph.D. thesis, Electrical Engineering Department, MIT.
- Feng, Z., & Hansen, E. A. (2001). Approximate planning for factored POMDPs. In Proceedings of the 6th European Conference on Planning (ECP-01).
- Feng, Z., & Hansen, E. A. (2004). An approach to state aggregation for POMDPs. In Proceedings of the 2004 AAAI workshop on learning and planning in Markov processes
  Advances and challenges., San Jose, CA.
- Feng, Z., & Zilberstein, S. (2004). Region-based incremental pruning for POMDPs. In Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence (UAI-2004).
- Feng, Z., & Zilberstein, S. (2005). Efficient maximization in solving pomdps. In Proceedings of the Twentieth of the Nineteenth National Conference on Artificial Intelligence (AAAI-05), Pittsburgh, PA.
- Hansen, E. (1998). Finite-memory control of partially observable systems. Ph.D. thesis, Department of Computer Science, University of Massachusetts at Amherst.
- Hansen, E., & Zhou, R. (2003). Synthesis of hierarchical finite-state controllers for POMDPs. In Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling.
- Hauskrecht, M. (1997). Incremental methods for computing bounds in partially observable markov decision processes. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97)*, Providence, RI.
- Hauskrecht, M. (2000). Value-function approximations for partially observable Markov decision processes. Journal of Artificial Intelligence Research, 13, 33–94.
- Ioslovich, I. (2001). Robust reduction of a class of large-scale linear programs. SIAM Journal on Optimization, 12(1), 262–282.

- Littman, M. L. (1994). The witness algorithm: Solving partially observable markov decision processes. Tech. rep. CS-94-40, Brown University Department of Computer Science.
- Littman, M. L., Cassandra, A. R., & Kaelbling, L. P. (1996). Efficient dynamic-programming updates in partially observable markov decision processes. Tech. rep. CS-95-19, Brown University, Providence, RI.
- Lovejoy, W. (1991). Computationally feasible bounds for partially observed markov decision processes. *Operations Research*, 39, 162–175.
- Ng, A. Y., & Jordan, M. I. (2000). PEGASUS: A policy search method for large MDPs and POMDPs. In *Proceedings of the 16th conference on Uncertainty in Artificial Intelligence.*
- Pineau, J., Gordon, G., & Thrun, S. (2003). Policy-contingent abstraction for robust robot control. In UAI-03.
- Poupart, P., & Boutilier, C. (2003). Bounded finite state controllers. In Proceedings of Advances in Neural Information Processing Systems 16.
- Roy, N., & Gordon, G. (2003). Exponential family PCA for belief compression in POMDPs. In Advances in Neural Information Processing 15.
- Roy, N., & Thrun, S. (1999). Coastal navigation with mobile robots. In Advances in Neural Information Processing 12.
- Smallwood, R. D., & Sondik, E. J. (1973). The optimal control of partially observable Markov processes over a finite horizon. Operations Research, 21(5), 1071–1088.
- Sondik, E. J. (1971). The optimal control of partially observable Markov processes. Ph.D. thesis, Stanford University.
- Sondik, E. J. (1978). The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. Operations Research, 26, 282–304.
- Theocharous, G., & Mahadevan, S. (2002). Approximate planning with hierarchical partially observable Markov decision processes for robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Washington, D.C.
- White, C. C. (1991). A survey of solution techniques for the partially observed markov decision process. Annals of Operations Research, 32, 215–230.
- Zhang, N. L., & Liu, W. (1996). Planning in stochastic domains: Problem characteristics and approximation. Tech. rep. HKUST-CS96-31, Hong Kong University of Science and Technology.
- Zhou, R., & Hansen, E. A. (2001). An improved grid-based approximation algorithm for POMDPs. In Proceedings of the 17th International Joint Conference on Artificial Intelligence.