Solving POMDPs from Both Sides: Growing Dual Parsimonious Bounds

Nicholas Armstrong-Crews
Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

Geoffrey Gordon
Machine Learning Dept.
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

Manuela Veloso
Computer Science Dept.
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

Abstract

Partially Observable Markov Decision Processes, or POMDPs, are useful for representing a variety of decision problems; unfortunately, solving for an optimal policy is computational intractable in general. In this paper, we present a set of novel search techniques for solving POMDPs approximately. We build on previous heuristic search and point-based algorithms, but improve upon them in several ways: we introduce an efficient method for approximating the convex hull of the upper bound, we expose embedded finite Markov structure in each bound, and we show how to prune aggressively while still maintaining convergence. The net result is a more targeted growth of the bound representations, leading to lower overall runtime and storage. We synthesize these contributions into a novel algorithm, which we call AAA-POMDP (Appropriately Acronymmed Algorithm). We describe its theoretical properties, including computational efficiency, and examine its performance on on standard benchmark problems from the literature. On these problems, our algorithms exhibit competitive or superior performance when compared to previous methods.

Partially Observable Markov Decision Processes (POMDPs) provide an elegant and general framework for representing decision problems with uncertain state and action outcomes (Sondik 1971) (Kaelbling, Littman, and Cassandra 1998), but unfortunately finding optimal (or ϵ-optimal) policies is computationally intractable, in general (Papadimitriou and Tsisiklis 1987). Recent advances have relied on using a finite set of belief points and expanding that set during the search (Pineau, Gordon, and Thrun 2003) (Smith and Simmons 2004) (Ji et al. 2007) (Spaan and Vlassis 2005). This approach helps in two ways: it reduces the beliefs considered from a continuous set to a finite set, a more computationally manageable task; and it provides intelligent heuristics to focus the policy search.

Historically, POMDP solvers have maintained a set of hyperplanes whose convex hull is a lower bound for the value function. These algorithms add new hyperplanes via Bellman backups and prune unnecessary hyperplanes. The lower-bounding value of a belief is its projection onto the lower-bounding convex hull, which can be computed efficiently by maximizing over its projection onto each hyperplane (a dot product calculation). HSVI (Smith and Simmons 2004) also maintains an upper bound, represented as the convex hull of a set of points; it adds new points by looking ahead via action and observation heuristics. The upper-bounding value of a belief is its projection onto the upper-bounding convex hull. Unfortunately, this value cannot be computed efficiently for non-vertices (it requires solving a linear program), which is especially problematic since nearly all HSVI subroutines rely on computing this value. HSVI2 uses a faster but less tight approximation to the upper-bounding convex hull.

POMDP policy iteration (Sondik 1971) (Hansen 1998) (Ji et al. 2007) is an alternative to performing Bellman backups on hyperplanes. It represents a policy as a finite-state controller (FSC); for a given initial controller state and POMDP world state, the FSC induces a Markov chain whose value can be computed efficiently. The algorithm proceeds by evaluating the chains for each controller state and world state, adding new controller states that will result in improved values, and pruning old controller states that have no predecessors. There are two aspects of this algorithm that chafe. Firstly, controller states are not modified once they've been added, even if it becomes clear that a different action would be optimal; instead, the algorithm waits until new states are added that allow pruning of the old state. Secondly, the algorithm only prunes controller states when it can guarantee that doing so will not reduce the value function *anywhere*, even if the the reduction would only occur in a non-reachable portion of the belief space.

Our contributions in this paper are threefold:

1. We present a novel method of maintaining the upper bound — to our knowledge, the first efficient, convergent point-based method to do so. By doing so, we address the inefficiency of maintaining an upper bound (the same issue that afflicts HSVI).
2. We show that POMDP policy iteration is in fact solving a finite-state MDP in a restricted policy class. We show how to construct the MDPs, for both upper and lower bounds, and how to solve them efficiently. This technique addresses the first stated problem with policy iteration.
3. We propose an improved pruning criterion that prunes more aggressively but still guarantees convergence, thereby solving the second stated problem with policy it-

eration.

We unify these contributions as a new POMDP solver, the AAA-POMDP (Appropriately Acronymmed Algorithm). AAA maintains upper and lower bounds on the value function while quickly converging to a value function whose representation uses little space (a "parsimonious" representation). We examine the algorithm's performance on several benchmark problems from the literature. In comparison to previous point-based schemes, we find AAA's performance to be competitive or superior.

Background

The MDP

The Markov Decision Process, or MDP, is a framework for modeling an agent moving stochastically among a finite set of states via probabilistic transitions influenced by the agent's choice of actions. The agent's next state is only dependent upon its previous state and its current choice of action. The agent receives a reward signal at each timestep, associated with the current state and action. In this paper, we consider the case of discounted reward and infinite horizon. The tuple $(\gamma, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$ specifies the MDP: γ is the scalar discount factor; $\mathcal{S}$ is the set of all states in the world; $\mathcal{A}$ is the set of all possible actions; $\mathcal{T} : \mathcal{S} \times \mathcal{A} \mapsto \Pi(\mathcal{S})$ is the state transition function, written $\mathcal{T}(s, a, s')$ for the transition s to s'; and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \mapsto \Re$ is the (immediate) reward function.

The objective of the agent is to maximize long-term expected reward $E[\sum_0^\infty \gamma^t r_t]$, where r_t is the immediate reward at time t associated with state s_t and action a_t at time t. The agent maximizes this expectation by determining a stationary policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ that completely determines the agent's behavior. The optimal policy can be generated by, for each state s, greedily choosing the action that maximizes $J(s)$, the long-term expected reward for s assuming the agent acts optimally (also called the value function and often denoted V).

The POMDP

The Partially Observable Markov Decision Process, or POMDP, extends the MDP framework to allow for incomplete knowledge of state. The agent now receives observations that are generated stochastically from a set of possible observations at each state. Rather than true state, the agent maintains in memory a belief state, which consists of a discrete probability distribution over states and is a sufficient statistic for its history to act optimally in the future. The POMDP is then Markovian in belief state, since all the agent requires to model the transition to next belief state is its current belief state and its action. The tuple for a POMDP, $(\gamma, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O})$, contains the same elements as an MDP, plus: Ω, the set of all possible observations; and $\mathcal{O} : \mathcal{S} \times \mathcal{A} \mapsto \Pi(\Omega)$, the observation function.

The agent still wishes to maximize long-term expected reward, and can do so by computing a stationary policy $\pi : \Delta \mapsto \mathcal{A}$, where Δ is the probability simplex $\Pi(\mathcal{S})$. An optimal policy induces an optimal value function, which should satisfy Bellman's Equation for POMDPs (1):

$$J(b) \equiv \max_a \left(\mathcal{R}(\cdot, a)^T b + \sum_o Pr(o|a, b) J(b_a^o) \right) \quad (1)$$

where

$$\begin{aligned} Pr(o|a, b) &= \sum_s b(s) Pr(o|a, s) \\ &= \mathbf{O}_a^o \mathbf{T}_a^T b. \end{aligned}$$

Here we use bold capitals as a shorthand for the matrices representing the POMDP tuple functions; subscripts and superscripts indicate indexes in the proper dimensions.

One challenge in finding the optimal value function is that Equation (1) has as its domain the continuous belief simplex Δ. It turns out that the value function is piecewise linear and convex for a finite horizon, and can be approximated to arbitrary precision by a piecewise linear and convex function in the infinite horizon. Such a function is typically represented as the max over a set of hyperplanes: $J(b) = \max_{\alpha \in \Gamma} b^T \alpha$.

Most solution algorithms attempt to maintain a parsimonious set Γ (one that contains relatively few vectors to represent the value function), in the interest of computational efficiency. Heuristic search (Smith and Simmons 2004), exact value iteration (Sondik 1971), randomization (Spaan and Vlassis 2005), and linear programming (Littman) have been applied with some success. Methods for belief compression, both lossy (Roy and Gordon 2002) and lossless (Poupart and Boutilier 2003), have been shown effective in reducing the problem size when an appropriate compression scheme is known.

Our Approach

First, we describe how we maintain bounds and perform backups. Next, we discuss the embedded finite Markov structure of the bound representations and how we can use it to speed up our search. Finally, we present our pruning methodology and show that it still converges while pruning more aggressively and thus maintaining a smaller representation.

Maintaining Bounds

We maintain the lower bound as a set of vectors $\underline{\alpha} \in \Gamma$. The value of a belief b using the lower bound is given by:

$$\underline{J}(b) = \max_{\underline{\alpha} \in \Gamma} b^T \underline{\alpha} \quad (2)$$

This is the standard represenation and is quick to compute, taking $O(|\Gamma||\mathcal{S}|)$ time.

We maintain the upper bound as a set of points and their values: $(b, z) \in (B, Z)$. As an upper bound, we have $J(b) \leq z$ for $b \in B$; and since we know the value function is convex, we can define the value function at the rest of the beliefs in the simplex $b' \in \Delta$ as the convex hull of $(b, z) \in (B, Z)$. Evaluating a new belief is a computationally intensive task, equivalent to projecting onto the convex

hull of (B, Z), or solving the linear program (LP):

$$\bar{J}(b) = \min_{w} \; z^T \, w \tag{3}$$

$$\text{subject to :}$$

$$\begin{aligned} \mathbf{B}w &= b \\ \sum_i w_i &= 1 \\ w_i &\geq \mathbf{0} \quad \forall \, i \end{aligned}$$

We can break the projection into two steps:

1. *Choose the vertices defining the face*. A set of $|\mathcal{S}|$ points $(b, z) \in (B_f, Z_f)$ defines a face (at most — less for degenerate faces). This set must encompass b; in other words, b must lie in its convex hull. Any set of points that encompasses b will induce a valid upper bound, though not necessarily the optimal one.
2. *Calculate the projection onto the face*. The equation of the hyperplane defining the face can be calculated by solving the linear system $\mathbf{B}_f \bar{\alpha} = \mathbf{Z}_f$.

We can find the equation of the face by performing the Moore-Penrose pseudo-inverse on the matrix of beliefs that defines the face's vertices:

$$w = \mathbf{B}_f^+ b = (\mathbf{B}_f^T \mathbf{B}_f)^{-1} \mathbf{B}_f^T b \tag{4}$$

We must still maintain that w be nonnegative. Our final equation for the upper bound, given a choice of vertices (B_f, Z_f), is then:

$$\bar{J}(b) = \mathbf{Z}_f^T \, \mathbf{B}_f^+ b, \tag{5}$$

which is a valid upper bound only when $\mathbf{B}_f^+ b \succeq 0$.

The Sharktooth Approximate Convex Hull

We can group terms in Equation (5) as $\mathbf{Z}_f^T (\mathbf{B}_f^+ b) = \mathbf{Z}_f^T w$ to retrieve our original LP. However, if we group the terms as $(\mathbf{Z}_f^T \mathbf{B}_f^+) b$, then we produce an equation linear in b:

$$\bar{J}(b) = \bar{\alpha}^T b \tag{6}$$

In the lower bound, we've described expressions of this form as α-vectors — indeed, that is precisely what we have now for the upper bound! Viewed another way, Equation (6) describes the dual of the upper bound LP: the minimal bounding hyperplane at b. As long as our restriction of $w \succeq 0$ holds, we can use $\bar{\alpha}$ to evaluate beliefs efficiently, avoiding the need to solve the full LP. While computing $\bar{\alpha}$ is still somewhat expensive, since it requires computing $\mathbf{B}_f^+$, we can use it to quickly evaluate many beliefs thereafter.

In other words, our basic strategy to reduce computation for evaluating the upper bound is to cache a set of faces F, each face $f \in F$ stored with its relevant properties: $(\mathbf{B}_f, \mathbf{Z}_f, mathbf B_f^+, \bar{\alpha}_f,)$. Since the number of faces grows as $|\mathcal{B}|$ choose $|\mathcal{S}|$, but we might never use many of these faces, we maintain only a small set F. The remainder of this section is concerned with how we store, update, and use this set.

Evaluating a Belief To evaluate a belief b, we find the subset of *qualified* faces $F_Q \subseteq F$ satisfying our $w \succeq 0$ restriction. This restriction is equivalent to requiring that the face contain the belief. We can then use the face that is minimal at b, and the belief's value is that minimum.

The matrix multiplication to obtain w is $O(|\mathcal{S}|^2)$. Evaluating the value of b on that face only requires $O(|\mathcal{S}|)$. Hence, it is wise to quickly eliminate faces that could not be valid at b, then minimize over the remaining faces. In general, any points that lie outside a simplex's circumsphere also lie outside the simplex. In higher dimensions the circumsphere becomes far too conservative, so we also test if the point lies outside the bounding box. Each of these tests requires $O(|\mathcal{S}|)$ computation, so faces they eliminate from consideration are a computational gain.

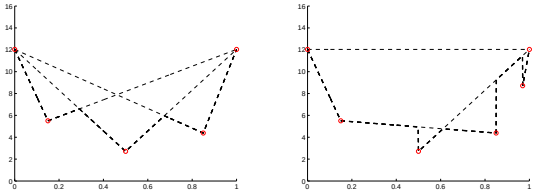
Alternatively, we could structure the search for containing faces as a KD-tree, reducing the set of potential containing faces at each branch; but for simplicity in implementation, we did not do so in this paper.

HSVI2 (Smith and Simmons 2005) uses a similar strategy to that we've described to approximate the convex hull: the so-called "sawtooth" approximation. It generates a set of valid faces characterized by taking all but one of the simplex corners and one of the interior belief points. For each interior belief, it is trivial to see which simplex corners are also needed in the face to contain b. Hence, it makes $|B| - |\mathcal{S}|$ faces, since B can always be partitioned into $|\mathcal{S}|$ corner beliefs and $|B| - |\mathcal{S}|$ interior beliefs. Computing each face can be viewed as a single rank-one update to the inverse of $\mathbf{B}_f$, which is, in this case, the identity. A rank-one update is computed in $O(|\mathcal{S}|^2)$ with the Sherman-Morris formula:

$$\begin{aligned} u &= I(i \stackrel{?}{=} k) \\ v &= b - b' \\ \left(\mathbf{B}_f + uv^T\right)^+ &= \mathbf{B}_f^+ \frac{\mathbf{B}_f^+ uv^T \mathbf{B}_f^+}{1 + v^T \mathbf{B}_f^+ u}, \end{aligned}$$

where u is a column vector whose entries are zeros except for row k, b' is the belief to be removed (the k'th belief in $\mathbf{B}_f$), and b is the point to be added. In the case of the sawtooth approximation, $\mathbf{B}_f^+$ is the identity, so the total cost is then $O(|B||\mathcal{S}|)$.

This practice begs the question: why a single rank-one update? Furthermore, the faces are reconstructed at each iteration. We introduce a generalized version of sawtooth, which we call the "sharktooth" approximation, that remembers the previous face and updates that face by rank-one updates. The name is in analogy with "sawtooth," and is characterized by multiple overlapping faces or "teeth." The distinction is depicted in Figure 1. In essence, the teeth are cached faces, preserved from previous iterations. The net result is that we interleave iterations of solving for the convex hull with belief expansion and value iteration steps, better targeting the faces whose convex hull influences the value function and policy. After a single update, the sharktooth and sawtooth bounds are identical; additional updates improve sharktooth further, converging to the true convex hull in the limit.



(a) Sawtooth (b) Sharktooth

Figure 1: The sawtooth (left) and sharktooth (right) approximations to the convex hull. Ignoring the near-vertical faces, which are artifacts of MATLAB plotting, one can see that the sharktooth approximation is tighter and uses less faces.

Algorithm 1 $f = \text{CHOOSEFACESHARKTOOTH}(b, F)$

1: Find qualified faces $F_Q \subseteq F$ that contain b in their bounding box and circumsphere.
2: Find valid faces $F_V \subseteq F_Q$ that contain b: $\{f \in F_Q | w = \mathbf{B}_f^+ b \succeq 0\}$.
3: Choose the minimal $f \leftarrow \arg\min_{f \in F_V} \overline{\alpha}_f^T b$

Improving Bounds

Let's say we'd like to improve our bounds at some point b, for which we've already computed an updated value z. In the lower bound, we simply perform a point-based vector backup, given in Algorithm 2. In the upper bound, we wish to add new faces or improve on prior faces. Adding new faces is described in the next section on performing backups; we now describe how to improve our prior faces in F. We can do what is essentially an iteration of the revised simplex method. For a face f witnessed by a point b_f, we consider swapping a vertex in (B_f, Z_f) for a point in (B, Z). Here we mean (B, Z) after including the new point (b, z); we must include old points as candidates because we run only a single iteration to improve the sharktooth bound, rather than iterating until it converges to the convex hull. We can compute in $O(|\mathcal{S}|)$ time whether this swap is feasible and would improve the value of b. If so, we perform the swap and compute the new hyperplane $\bar{\alpha}$ via a rank-one update.

Performing this improvement step for each face would require $O(\ |B|\ (|\mathcal{S}|^2 + |F||\mathcal{S}|)\ )$ time.

Point-based Backups

There are two basic steps in performing a backup (for either the upper or lower bound): 1) choose the successor faces for all actions and observations; and 2) evaluate Bellman's equation using those faces. It is clear that we can back up a point in this fashion; and since fixing the action and successor elements in Bellman's equation makes it linear in b, we can back up a vector similarly.

Point-based algorithms back up vectors (in the lower bound) by using successor faces that are maximal at b_o^a. We also back up vectors in the upper bound, by choosing successor faces with CHOOSEFACESHARKTOOTH.

Algorithm 2 $\alpha = \text{POINTBASEDVECTORBACKUP}(b, \Gamma)$

1: $f(a, o) \leftarrow \text{CHOOSEFACE}(b_o^a)$
2: $\beta_a^o \leftarrow \Gamma_{f(a,o)}$
3: $\beta_a \leftarrow \mathbf{R}_a + \gamma \sum_o \mathbf{T}_a \mathbf{D}_a^o \beta_a^o$
4: $J(b) \leftarrow \max_a \beta_a^T b$
5: $\alpha \leftarrow \arg\max_a \beta_a^T b$

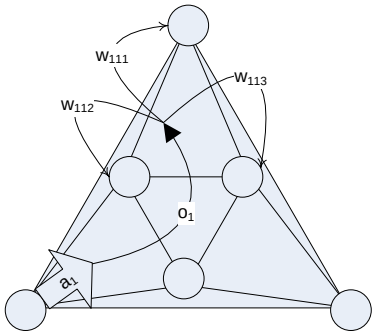


Figure 2: An "overhead" view of the Upper Bound MDP for a 3-state POMDP, looking down the value function axis. MDP states are vertices of the convex hull, shown here as circles. Faces of the convex hull are shown as sub-triangles of the triangular simplex. Action a_1 results in beliefs that lie outside our set B; the transitions w move us back onto B.

The method for point-based vector backups is summarized in Algorithm 2. Note that it applies to both upper and lower bounds; in line 1, CHOOSEFACE means CHOOSEFACESHARKTOOTH for the upper bound, while for the lower bound it indicates the maximal vector at b.

Embedded Markov Structure

Upper Bound MDP Substituting Equation (5) into the right-hand size of Equation (1), and expanding b_a^o, we get:

$$\bar{J}(b) = \max_a b^T \left(\mathbf{R}_a + \gamma \sum_o \mathbf{T}_a \mathbf{D}_a^o \mathbf{B}_{f(a,o)}^{+T} \mathbf{Z}_{f(a,o)} \right), \tag{7}$$

where $f(a, o)$ is the index of the face that is maximal for b_a^o and $\mathbf{D}_a^o$ is shorthand for $\text{DIAG}(\mathbf{O}_a^o)$.

Recalling that (B, Z) is a finite set, it is easy to see that the above equation defines a finite-state MDP, with: B as the set of states; $\mathbf{B}^T \mathbf{R}_a$ as the reward vector for action a; and $\mathbf{T}_a \mathbf{D}_a^o \mathbf{B}_{f(a,o)}^{+T}$ for the transition matrix. This transition matrix can be interpreted as a state transition, as in a normal MDP; followed by an observation, modifying belief state as in the belief MDP; and finally, a "bonus" observation, moving b_a^o to a face vertex i with probability w_i. [1] This final step ensures that transitions are made on the closed set B.

A POMDP over a finite set of states can be reduced to a continuous MDP in belief state. Hence, it is not surprising

[1] The "bonus observation" description is merely an intuitive interpretation of the MDP transition matrix, *not* an explicit mechanism of the system.

that considering a finite subset of the belief points yields a discrete MDP.

Performing backups, as in HSVI, solves this MDP implicitly by value iteration (each backup is one step of value iteration at one point); however, it does not solve it to convergence. Instead, we explicitly perform value iteration steps to convergence with a fixed set (B, Z). While this method is more expensive computationally, it will give us better improvements in the upper bound for fewer points. Furthermore, we expect the values to change little when we add new points, so we seed value iteration with the previous values Z, and convergence is typically quite fast.

Lower Bound MDP Finite Markov structure was earlier noted in policy iteration methods (Hansen 1998); given a finite-state controller (FSC) representing a policy, Equation (1) defines a Markov chain over $\mathcal{S} \times \Gamma$, where $\mathcal{S}$ is the original state space and Γ comprises the states of the FSC.

It is also the case that in an MDP, fixing a policy yields a Markov chain. It is a natural step to try to "upgrade" the chain to an MDP by allowing actions to vary during the solution. However, if we allow action selection at each state in $\mathcal{S} \times \Gamma$, we are essentially allowing knowledge of the underlying system state $s \in \mathcal{S}$. Instead, we must restrict our policy class to a single action across all $s \in \mathcal{S}$ for a fixed $\alpha \in \Gamma$. We do so by using a representative belief point for each vector — specifically, the point used to generate the vector from POINTBASEDVECTORBACKUP — to select the action for the vector. We interleave this policy improvement step with a policy evaluation step, performed by value iteration. Iterating the two steps together yields a policy iteration algorithm, but one that is different from prior work (Hansen 1998) because it solves for a policy of fixed size. It is *not* guaranteed to improve policy values at each iteration, but overall we find that it gives a better policy than not iterating. Since this is a subroutine producing Γ', we achieve monotonicity in the policy by integrating it into our prior policy: $\Gamma \leftarrow \text{MERGE}(\Gamma, \Gamma')$. The MERGE procedure is described in prior work on policy iteration (Hansen 1998) (Ji et al. 2007).

Finally, we prune any vectors (and their associated controller states) that are not witnessed by any belief that we've visited before (in EXPLORE), or will visit in the next step $\bigcup_{a,o}\{b_a^o\}$. This last step allows pruning of Γ', something prior policy iteration algorithms could not do, while still guaranteeing convergence (see theoretical results in Section). Note that we expect this policy iteration to converge quickly, since we seed it with the prior policy Γ; similarly, we expect policy evaluation via value iteration to converge quickly, since we seed it with the value of the representative belief for each vector $b^T\alpha$.

Pruning

"Pruning" refers to removing unnecessary points or vectors (faces) from our bounds. We say a vector α is *unnecessary* if it has no witness point b such that $\alpha \cdot b = J(b)$ (otherwise, we say b *witnesses* α); we say a point (b, z) is unnecessary if it is not a vertex of the convex hull. Most algorithms prune conservatively, requiring that the pruning does not loosen the bounds nor invalidate them at *any* $b \in \Delta$. In other words, these algorithms guarantee *monotonicity* and *validity*. An exception is PBVI, which prunes (implicitly) all vectors not witnessed by some $b \in B$. This practice does not guarantee monotonicity, not even for $b \in B$ (Pineau, Gordon, and Thrun 2003).

We adopt an approach not as aggressive as PBVI nor as conservative as other algorithms: specifically, we prune vectors (faces) that are not witnessed by any belief we've previously visited nor might visit with one step of lookahead. We argue this pruning operation is *one-step monotone*, in the sense that it will loosen the bounds at no previously examined belief nor any to be examined in the next round of backups. Since we will perform a backup to add new vectors at the next iteration of the algorithm, the pruning operation *must* be one-step monotone to guarantee that our backed-up values won't result in loosened bounds at $b \in B$.

Points in the upper bound are implicitly pruned as a result of pruning faces; since we construct the set of vertices as $\bigcup_f (B_f, Z_f)$, a vertex will no longer appear in this set if all faces it supports are removed.

Where to Improve

A backup could be performed at any arbitrary $b \in \Delta$ (or at all of them simultaneously, as in a full functional backup). However, techniques for partial backups based on heuristic lookahead have seen great success. As such, we adopt HSVI's lookahead heuristics; namely, we choose to descend the search path with: 1) the action whose value is maximal in the upper bound, and 2) the observation that contributes most to the bound width at b. We stop exploring when the discounted bound width is below ϵ. The complete AAA algorithm is listed as Algorithm 4.

Algorithm 3 $(B', Z') = \text{EXPLORE}(b, \epsilon, t)$

if $\bar{J}(b) - \underline{J}(b) \geq \epsilon\gamma^{-t}$ **then**
  $a^* \leftarrow \arg\max_a \bar{Q}(b, a)$
  $o^* \leftarrow \arg\max_o Pr(o|b, a)\psi(b_a^o)$
  $(B', Z') \leftarrow \text{EXPLORE}(b_a^o, \epsilon, t+1)$
  $(B', Z') \leftarrow (B', Z') \bigcup \{b, \bar{Q}(b, a^*)\}$
end if

Algorithm 4 $\text{AAA}(F, \Gamma)$

Initialize Γ by evaluating any initial policy.
Initialize F with a single face, consisting of the simplex corners and the corresponding MDP value for that state.
while $\bar{J}(b_0) - \underline{J}(b_0) > \epsilon$ **do**
  $(B', z') \leftarrow \text{EXPLORE}(b_0, \epsilon, 0)$
  $(F, \Gamma) \leftarrow \text{IMPROVEBOUNDS}(B', z')$
  $(F, \Gamma') \leftarrow \text{SOLVEEMBEDDEDMDPS}(F, \Gamma)$
  $\Gamma \leftarrow \text{MERGE}(\Gamma, \Gamma')$
  $(F, \Gamma) \leftarrow \text{PRUNE}(F, \Gamma)$
end while

Convergence

We now summarize theoretical results regarding convergence and runtime.

Lemma 1. EXPLORE *is monotone, in the sense that visiting a belief point b at iteration t_1 and $t_2 > t_1$ implies $\bar{J}^{t_1}(b) \geq \bar{J}^{t_2}(b)$ and $\underline{J}^{t_1}(b) \leq \underline{J}^{t_2}(b)$.*

The following results two are inherited from HSVI (Smith and Simmons 2004), since we use its search strategy and lookahead heuristics.

Lemma 2. *The search tree for* EXPLORE$(b_0, \epsilon, 0)$ *has finite depth:* $t_{\max} = \left\lceil \log_\gamma \frac{\epsilon}{||\bar{J}^0 - \underline{J}^0||_\infty} \right\rceil$.

Theorem 3. *Both $\bar{J}$ and $\underline{J}$ converge to within ϵ of the optimal value function, within $O(\ t_{\max}\ (|\mathcal{A}||\mathcal{O}|)^{t_{\max}}\ )$ time.*

While we inherit the worst-case number of iterations of HSVI, each iteration is faster (not requiring solving LPs directly). Furthermore, empirically the policy evaluation step improves results dramatically, especially for very small ϵ (Ji et al. 2007).

Theorem 4. *The size of the representation is bounded quadratically in $t_{\max}$; both $|\Gamma|$ and $|F|$ are $O(\ |\mathcal{S}||\mathcal{A}||\mathcal{O}|\ (t_{\max}^2 + |\mathcal{S}|)\ )$.*

Results

We now examine the performance of AAA on standard POMDP benchmark problems from the literature.

We implemented our own versions of PBVI, PBPI, and HSVI, since they share many of the same subroutines as our method. Moreover, using common implementations of these subroutines precludes implementation-specific differences, thus allowing direct performance comparisons.

We implemented all code in MATLAB, vectorizing it when possible but not painstakingly. All experiments were run on a 3.4 GHz dual-core Pentium IV machine with 2GB RAM. We ran each algorithm until it achieved empirical accumulated reward $\widehat{J}(b_0)$ (via 100 simulation episodes) was within $\epsilon = 10^{-3}$ of the optimal value $J(b_0)$ (as given by the converged bounds $[\underline{J}(b_0), \bar{J}(b_0)]$ of AAA and HSVI).

Table 1: Results on Benchmark Problems

	PBVI	**PBPI**	**HSVI**	**AAA**
Maze (20s,6a,8o)				
Runtime (s)	57	.49	50	.25
$\|\Gamma\|$	135	7	17	6
Hallway (57s,5a,21o)				
Runtime (s)	324	163	3347	87
$\|\Gamma\|$	94	194	1252	51

Additionally, it is instructive to examine the tightness of the bounds as the algorithms progress; although showing through simulation that they have achieved a near-optimal solution, the bounds constitute a proof that they are within ϵ, a much stronger statement. In our experiments, PBVI and PBPI's bounds did not change from their initial values by more than an order of magnitude (their bounds depend on approaching the reachable belief space $\bar{\Delta}$ in the limit, and in our experiments, we do not even come close to this limiting case). Restricting our comparison to HSVI, then, we find that AAA truly shines; in a given number of iterations, it can achieve several orders of magnitude smaller bound width than HSVI. The result is even better when we compare using wallclock time, since AAA's iterations are faster than HSVI's. For example, in the Hallway problem, HSVI requires 3347 seconds to achieve a bound width of 10^{-2}, after which time AAA has achieved a bound width of 10^{-6}. We ascribe the jump in performance to the fact that solving the MDP via many cheap MDP value iteration steps is much faster than perfoming POMDP backups (and $|\mathcal{A}||\mathcal{O}|$ expensive belief evaluations for each backup).

Conclusions and Future Work

In this paper, we contribute an efficient method for maintaining an approximate upper-bounding convex hull, a method for identifying and solving MDPs in the upper and lower bounds, and a pruning strategy that is efficient, aggressive, and does not break convergence. Initial results on benchmark problems are encouraging, but more extensive experiments would be desirable. Intelligent data structures (such as KD-trees) and caching could speed up many processes, as well as a C++ implementation. We envision a second implementation, not unlike HSVI2, which benefits from these and other tricks. On the theoretical front, we plan to investigate duality of the bounds in further detail.

References

Hansen, E. 1998. An improved policy iteration algorithm for POMDPs. In *Proceedings of NIPS-98*.

Ji, S.; Parr, R.; Li, H.; Liao, X.; and Carin, L. 2007. Point-based policy iteration.

Kaelbling, L. P.; Littman, M.; and Cassandra, A. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence Journal* 101(1-2):99–134.

Littman, M. The witness algorithm: Solving POMDPs. Technical report.

Papadimitriou, C., and Tsisiklis, J. 1987. The complexity of markov decision processes. *Mathematics of Operations Research* 12(3):441–450.

Pineau, J.; Gordon, G.; and Thrun, S. 2003. Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of IJCAI-03*.

Poupart, P., and Boutilier, C. 2003. Value-directed compression of POMDPs. In *Proceedings of NIPS-03*.

Roy, N., and Gordon, G. 2002. Exp. family PCA for belief compression in POMDPs. In *Proceedings of NIPS-02*.

Smith, T., and Simmons, R. 2004. Heuristic search value iteration for POMDPs. In *Proceedings of UAI-04*.

Smith, T., and Simmons, R. 2005. Point-based POMDP algorithms: Improved analysis and implementation. In *Proceedings of UAI-05*.

Sondik, E. 1971. *The Optimal Control of Partially Observable Markov Processes*. Ph.D. Dissertation, Stanford.

Spaan, M., and Vlassis, N. 2005. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research* 24.