Point-Based Policy Iteration

Shihao Ji, Ronald Parr[†], Hui Li, Xuejun Liao, and **Lawrence Carin**
Department of Electrical and Computer Engineering
[†]Department of Computer Science
Duke University
Durham, NC 27708-0291

Abstract

We describe a point-based policy iteration (PBPI) algorithm for infinite-horizon POMDPs. PBPI replaces the exact policy improvement step of Hansen's policy iteration with point-based value iteration (PBVI). Despite being an approximate algorithm, PBPI is monotonic: At each iteration before convergence, PBPI produces a policy for which the values increase for at least one of a finite set of initial belief states, and decrease for none of these states. In contrast, PBVI cannot guarantee monotonic improvement of the value function or the policy. In practice PBPI generally needs a lower density of point coverage in the simplex and tends to produce superior policies with less computation. Experiments on several benchmark problems (up to 12,545 states) demonstrate the scalability and robustness of the PBPI algorithm.

Introduction

Point based algorithms, such as PBVI (Pineau, Gordon, & Thrun 2003) and Perseus (Spaan & Vlassis 2005), have become popular in recent years as methods for approximating POMDP policies. Point based algorithms have the computational advantage of approximating the value function only at a finite set of belief points. This permits much faster updates of the value function compared to exact methods, such as the witness algorithm (Kaelbling, Littman, & Cassandra 1998), or incremental pruning (Cassandra, Littman, & Zhang 1997), which consider the entire belief simplex.

While it is possible to establish error bounds for point based algorithms based upon the density of point coverage in the belief simplex, such results are typically too loose to be of practical significance. For large state-space POMDP problems, it is often infeasible to maintain a high density of point coverage over the entire belief simplex. Thus, coarsely sampled belief points in high-dimensional space can often result in large approximation error relative to the true value function.

In this paper we propose a point-based policy iteration (PBPI) algorithm. By replacing the exact policy improvement step of Hansen's policy iteration (Hansen 1998) with PBVI (Pineau, Gordon, & Thrun 2003), PBPI integrates the fast convergence of policy iteration and the high efficiency of PBVI for policy improvement. The resulting PBPI algorithm typically requires fewer iterations (each iteration includes a policy evaluation step and a PBVI policy improvement step) to achieve convergence *vis-à-vis* PBVI alone. Moreover, PBPI is monotonic: At each iteration before convergence, PBPI produces a policy for which the values increase for at least one of a finite set of initial belief states, and decrease for none of these states. In contrast, PBVI cannot guarantee monotonic improvement of the value function or the policy. In practice PBPI generally needs a lower density of point coverage in the simplex and tends to produce superior policies with less computation. We also discuss two variants of PBPI in which some of the properties of PBPI are relaxed. This allows a further study of the impact of monotonicity and policy evaluation on the performance of POMDP algorithms.

POMDP Review

Partially observable Markov decision processes (POMDPs) provide a rigorous mathematical model for planning under uncertainty (Smallwood & Sondik 1973; Sondik 1978; Kaelbling, Littman, & Cassandra 1998). A POMDP is defined by a set of states S, a set of actions A, and a set of observations Z. At each discrete time step, the environment is in some state $s \in S$; an agent takes action $a \in A$ from which it derives an expected reward $R(s, a)$. As a consequence, the environment transits to state $s' \in S$ with probability $p(s'|s, a)$, and the agent observes $z \in Z$ with probability $p(z|s', a)$. The goal of POMDP planning is to find a policy that, based upon the previous sequence of actions and observations, defines the optimal next action, with the goal of maximizing the discounted reward over a specified horizon. In this paper we consider an infinite horizon $\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)$, where $\gamma \in [0, 1)$ is a discount factor.

While the states cannot be observed directly, if the agent has access to the (correct) underlying model, it can maintain an internal belief state for optimal action selection. A belief state, denoted b, is a probability distribution over the finite set of states, S, with $b(s)$ representing the probability that the environment is currently in state s and $\sum_{s \in S} b(s) = 1$. It is well known that the belief state is a sufficient statistic for a given history of actions and observations (Smallwood & Sondik 1973), and it is updated at each time step by in-

corporating the latest action and observation via Bayes rule:

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

where b_a^z denotes the belief state updated from b by taking action a and observing z.

The dynamic behavior of the belief state is itself a discrete-time continuous-state Markov process (Smallwood & Sondik 1973), and a POMDP can be recast as a completely observable MDP with a $(|S|-1)$-dimensional continuous state space $\triangle$. Based on these facts, several exact algorithms (Kaelbling, Littman, & Cassandra 1998; Cassandra, Littman, & Zhang 1997) have been developed. However, because of exponential worst cast complexity, these algorithms typically are limited to solving problems with low tens of states. The poor scalability of exact algorithms has led to the development of a wide variety of approximate techniques (Pineau, Gordon, & Thrun 2003; Poupart & Boutilier 2003; Smith & Simmons 2005; Spaan & Vlassis 2005), of which PBVI (Pineau, Gordon, & Thrun 2003) proves to be a particularly simple and practical algorithm.

Point-Based Value Iteration

Instead of planning on the entire belief simplex $\triangle$, as exact value iteration does, point-based algorithms (Pineau, Gordon, & Thrun 2003; Spaan & Vlassis 2005) alleviate the computational load by planning only on a finite set of belief points B. They utilize the fact that most practical POMDP problems assume an initial belief b_0, and concentrate planning resources on regions of the simplex that are reachable (in simulation) from b_0. Based on this idea, Pineau *et al.* (2003) proposed a PBVI algorithm that first collects a finite set of belief points B by forward simulating the POMDP model and then maintains a single gradient (α-vector, in POMDP terminology) for each $b \in B$. This is summarized in Algorithm 1 (without the last two lines).

Algorithm 1. Point-based backup

function $\Gamma' = \text{backup}(\Gamma, B)$
% Γ is a set of α-vectors representing value function
$\Gamma' = \emptyset$
for each $b \in B$
  $\alpha_a^z = \arg\max_{\alpha\in\Gamma} \alpha \cdot b_a^z$, for every $a \in A$, $z \in Z$
  $\alpha_a(s) = R(s,a) + \gamma \sum_{z,s'} p(s'|s,a)p(z|s',a)\alpha_a^z(s')$,
  $\alpha' = \arg\max_{\{\alpha_a\}_{a\in A}} \alpha_a \cdot b$
  if $\alpha' \notin \Gamma'$, **then** $\Gamma' \leftarrow \Gamma' + \alpha'$, **end**
end
% The following two lines are added for modified backup
$\Lambda =$ all $\alpha \in \Gamma$ that are dominant at least at one $b_{\pi(b)}^z$
$\Gamma' = \Gamma' + \Lambda$

Hansen's Policy Iteration

Policy iteration (Sondik 1978; Hansen 1998) iterates over policies directly, in contrast to the indirect policy representation of value iteration. It consists of two interacting processes, one computing the value function of the current policy (policy evaluation), and the other computing an improved policy with respect to the current value function (policy improvement). These two processes alternate, until convergence is achieved (close) to an optimal policy.

Policy Evaluation Hansen (1998) represents the policy in the form of a finite-state controller (FSC), denoted by $\pi = \langle \mathcal{N}, \mathcal{E} \rangle$, where $\mathcal{N}$ denotes a finite set of nodes (or machine states) and $\mathcal{E}$ denotes a finite set of edges. Each machine state $n \in \mathcal{N}$ is labeled by an action $a \in A$, each edge $e \in \mathcal{E}$ by an observation $z \in Z$, and each machine state has one outward edge per observation. Consequently, a policy represented in this way can be executed by taking the action associated with the "current machine state", and changing the current machine state by following the edge labeled by the observation made.

As noted by Sondik (1978), the cross-product of the environment states S and machine states $\mathcal{N}$ constitutes a finite Markov chain, and the value function of policy π (represented by a set of α-vectors, with one vector corresponding to one machine state) can be calculated by solving the following system of linear equations:

$$\alpha_n^\pi(s) = R(s,a(n)) + \gamma \sum_{s',z} p(s'|s,a(n))p(z|s',a(n))\alpha_{l(n,z)}^\pi(s') \quad (2)$$

where $n \in \mathcal{N}$ is the index of a machine state, $a(n)$ is the action associated with machine state n, and $l(n,z)$ is the index of its successor machine state if z is observed. As a result, the value function of π can be represented by a set of $|\mathcal{N}|$ vectors $\Gamma_\pi = \{\alpha_1^\pi, \alpha_2^\pi, \cdots, \alpha_{|\mathcal{N}|}^\pi\}$, such that

$$V^\pi(b) = \max_{\alpha\in\Gamma_\pi} \alpha \cdot b$$

Policy Improvement The policy improvement step of Hansen's policy iteration involves dynamic programming to transform the value function V^π represented by Γ_π into an improved value function represented by another set of α-vectors, $\Gamma_{\pi'}$. As noted by Hansen (1998), each α-vector in $\Gamma_{\pi'}$ has a corresponding choice of action, and for each possible observation choice of an α-vector in Γ_π. This information can be used to transform an old FSC π into an improved FSC π' by a simple comparison of Γ_π and $\Gamma_{\pi'}$. A detailed procedure for FSC transformation is presented in Algorithm 2 when we introduce the PBPI algorithm, since PBPI shares the structure of Hansen's algorithm and has the same procedure for FSC transformation.

Point-Based Policy Iteration

Our point-based policy iteration (PBPI) algorithm aims to combine some of the most desirable properties of Hansen's policy iteration with point-based value iteration. Specifically, PBPI replaces the exact policy improvement step of Hansen's algorithm with PBVI (Pineau, Gordon, & Thrun 2003), such that policy improvement is concentrates on a finite sample beliefs B. This algorithm is summarized below in Algorithm 2, with the principal structure shared with Hansen's algorithm. There are two important differences between PBPI and Hansen's algorithm: (1) In PBPI, the backup operation only applies to the points in B, not the

entire simplex, which means that (2) the final pruning step can remove machine states that are unreachable for starting belief states not in B.

Algorithm 2. Point-based policy iteration

function $\pi = \text{PBPI}(\pi_0, B)$
% π_0 is an initial finite-state controller
$\pi' = \pi = \pi_0$;
do forever
  % Policy Evaluation
  Compute $\Gamma_\pi = \{\alpha_1^\pi, \alpha_2^\pi, \cdots, \alpha_{|\Gamma_\pi|}^\pi\}$ that represents the value function of π by (2);
  % Policy Improvement
  $\Gamma'_\pi = \text{backup}(\Gamma_\pi, B)$;
  % FSC Transformation: $\pi \rightarrow \pi'$
  for each $\alpha' \in \Gamma'_\pi$
    i. If the action and successor links associated with α' are the same as those of a machine state already in π, then keep that machine state unchanged in π';
    ii. Else if the vector α' pointwise dominates an α associated with a machine state of π, change the action and successor links of that machine state to those that correspond to α' (If α' pointwise dominates the α-vectors of more than one machine state, they can be combined into a single machine state.);
    iii. Else add a machine state to π' that has the action and successor links associated with α';
  end
  Prune any machine state for which there is no corresponding vector in Γ', as long as it is not reachable from a machine state to which an α-vector in Γ' does correspond;
  $\pi = \pi'$;
  if $\frac{1}{|B|}\sum_{b\in B} V^\pi(b)$ converges, **then** return π, **end**
end

Selection of Belief Set B The selection of a finite set of belief points B is crucial to the solution quality of PBVI and PBPI. Both algorithms generate the sample belief set by forward simulating the POMDP model. Let $B_0 = \{b_0\}$ be the set of initial belief points at time $t = 0$. For time $t = 1, 2, \cdots$, let B_t be the set of all possible b_a^z produced by (1), $\forall\ b \in B_{t-1}$, $\forall\ a \in A$, $\forall\ z \in Z$, such that $p(z|b,a) > 0$. Then $\cup_{t=0}^{\infty} B_t$, denoted $\bar{\triangle}$, is the set of belief points reachable by the POMDP. It is therefore sufficient to plan only on these reachable beliefs in order to find an optimal policy customized for the agent that starts from any $b \in \bar{\triangle}$, since $\bar{\triangle}$ constitutes a closed inter-transitioning belief set.

The reachable belief set $\bar{\triangle}$ may still be infinitely large. We thus obtain a manageable belief set by sampling $\bar{\triangle}$ with a heuristic technique that encourages sample spacing. We recursively expand the belief set B via the following procedure, starting with the initial belief set $B = \{b_0\}$. For every $b \in B$, we draw a sample z according to $p(z|b,a)$ for every $a \in A$. We thus obtain $|A|$ new belief points b_a, each generated by a different a. We select a single belief b_a^* from these new beliefs that has the maximum L_1 distance to the *current* B and add it into B only if its L_1 distance is beyond a given threshold ϵ.

The above procedure is similar to the one introduced by Pineau *et al.* (2003) when $\epsilon = 0$. When $\epsilon = 0$ the sample belief set B can grow so quickly that it often collects mostly belief points that are only a few simulation steps away from b_0. A larger value of ϵ tends to collect belief points that are further away from b_0 and yields a more uniform point coverage over the reachable belief set $\bar{\triangle}$.

Convergence and Error Bounds The proposed PBPI algorithm, despite being an approximate method, inherits many desirable properties of Hansen's policy iteration and point-based value iteration. For example, each iteration of PBPI adds at most $|B|$ new machine states to an FSC, as compared to $|A||\mathcal{N}|^{|Z|}$ that could be produced by Hansen's algorithm. In addition, we provide the following set of theorems to address PBPI's other properties.

Theorem 1. *At each iteration before convergence, PBPI produces a policy for which the values increase for at least one $b \in B$ and decreases for no $b \in B$, while this is not guaranteed for PBVI.*

Proof. We consider Algorithm 2 in four steps: (1) policy evaluation, (2) backup, (3) policy improvement, and (4) pruning. In step (1) policy evaluation computes the exact value of the policy π. The backup step (2) can be viewed as introducing a new set of machine states with corresponding α-vectors stored in Γ'_π. By construction, each new state and corresponding α-vector represent an action choice followed by an observation-conditional transition to machine states in π. Moreover, each $\alpha \in \Gamma'_\pi$ is the optimal such choice for some $b \in B$, given π. The policy improvement step (3) transforms states from π to π' by replacing states in π that are strictly dominated by the new states generated by the backup. Finally, the pruning in step (4) eliminates states which are not reachable from some $b \in B$ and, therefore, cannot cause a reduction in the value of starting in these states.

PBVI maintains only a single α-vector for each $b \in B$, and the value function of PBVI may decrease at some belief region after each point-based backup (see Fig. 1(b)). Further, because of the non-uniform improvement of the value function, PBVI may actually have reduced values at some $b \in B$. This arises when b transits to a b_a^z that happens to be in the belief region that has the reduced value (see Fig. 1(b)). This is likely to happen in practice, especially when the sample belief set is very sparse in the belief simplex. □

The original PBVI paper (Pineau, Gordon, & Thrun 2003) introduced the notion of a *pruning error* as a way of analyzing the error introduced by PBVI's failure to generate a full set of α-vectors. (Vectors not generated are implicitly pruned.) The pruning error measures the suboptimality of PBVI in comparison to a complete POMDP value iteration step. To understand the difference between the implicit pruning steps done by these algorithms, we ignore the policy evaluation step of PBPI, and focus on the

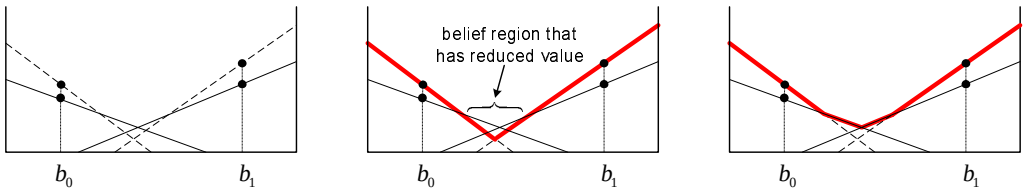

Figure 1: Example value functions of PBVI and PBPI. (a) point-based backup, where the solid lines represent the original α-vectors, and the dash lines represent the backed-up α-vectors; (b) the PBVI backed-up value function (solid thick lines); (c) the PBPI backed-up value function (solid thick lines).

backup and policy improvement steps, assuming that both algorithms start with the same Γ as input. PBVI computes $\Gamma^{PBVI} = \text{backup}(\Gamma, B)$ and discards Γ, while PBPI uses $\Gamma^{PBPI} = \Gamma^{PBVI} \cup \Gamma$ in its policy improvement step. Thus PBPI, can be viewed as drawing upon a larger set of α-vectors, and having less implicit pruning. This analysis only shows that PBPI does less implicit pruning than PBVI; it does not quantify the effects of the policy evaluation step. Note that the pruning implicit in the failure to do a complete value iteration step is different from the pruning of inferior or unreachable FSC states in Algorithm 2.

Theorem 2. *The pruning error introduced in PBPI is no larger smaller than that of PBVI and is bounded by:* $\eta^{PBPI}_{prune} \leq \eta^{PBVI}_{prune} \leq \frac{R_{max}-R_{min}}{1-\gamma}\varepsilon_B$, *where* ε_B *is the maximum distance from any* $b \in \bar{\triangle}$ *to* B.

Proof. Let $b' \in \bar{\triangle}$ be the point where PBVI makes its worst pruning error, and α' (generated by full value iteration) be the vector that is maximal at b'. Let Γ^{PBVI} and Γ^{PBPI} be the set of α-vectors whose upper surfaces form the PBVI and PBPI value functions, respectively. By erroneously pruning α', PBVI makes an error of at most $\eta^{PBVI}_{prune} = \alpha' \cdot b' - \max_{\alpha \in \Gamma^{PBVI}} \alpha \cdot b'$ and PBPI makes an error of at most $\eta^{PBPI}_{prune} = \alpha' \cdot b' - \max_{\alpha \in \Gamma^{PBPI}} \alpha \cdot b'$. So,

$$\begin{aligned}
\eta^{PBPI}_{prune} &= \alpha' \cdot b' - \max_{\alpha \in \Gamma^{PBPI}} \alpha \cdot b' \\
&\leq \alpha' \cdot b' - \max_{\alpha \in \Gamma^{PBVI}} \alpha \cdot b', \qquad \Gamma^{PBVI} \subset \Gamma^{PBPI} \\
&= \eta^{PBVI}_{prune} \leq \frac{R_{max} - R_{min}}{1-\gamma}\varepsilon_B
\end{aligned}$$

The last inequality is from Pineau *et al.* (2003). □

Algorithm Complexity It is known that PBVI has a time complexity of $\mathcal{O}(|B||S||A||Z||\Gamma_{t-1}|)$ for the t-th iteration (Pineau, Gordon, & Thrun 2003). In the case of the PBVI algorithm, the size of Γ_{t-1} is bounded by $|B|$. Thus, PBVI has a constant time complexity of $\mathcal{O}(|B|^2|S||A||Z|)$ at each iteration. But for PBPI each iteration could add at most $|B|$ new machine states to an FSC, and the size of Γ_{t-1} could be up to $(t-1)|B|$. Thus, the policy improvement step of PBPI has a worst-case time complexity of $\mathcal{O}\big(t|B|^2|S||A||Z|\big)$, which is linear with respect to the number of iterations. In addition, PBPI has the policy evaluation step and the FSC transformation step that costs extra time over PBVI. The policy evaluation step could potentially be quite expensive, as it requires solving a large system of linear equations. We take advantage of two facts about the system solved by PBPI to avoid this expense: (1) The system is sparse and (2) The values of many FSC states are often close to their values at the previous iteration of policy iteration. In such cases, it can be advantageous to solve the system indirectly via iterative updates. If the maximum out degree of the FSC states is S_o and the number of iterations to achieve acceptable precision is bounded by k, then the computational complexity for the indirect policy evaluation step is $\mathcal{O}(kt|B||S|S_o|Z|)$, where $S_o|Z|$ is the maximum number of non-zero coefficients for any variable in the linear system. Policy iteration with an indirect evaluation step is often termed modified policy iteration (Puterman & Shin 1978).

As demonstrated in the experiments that follow, PBPI's extra cost per iteration is compensated for in two ways. First, PBPI typically requires a much smaller B to achieve good performance than does PBVI. Second, PBPI can require fewer iterations to achieve convergence relative to PBVI. Thus, when initialized with a smaller belief set than PBVI, PBPI typically has a favorable performance profile, achieving higher quality policies with less computation.

Algorithm Variants

To further study of the impact of monotonicity and policy evaluation on the performance of POMDP algorithms, we relax some of the properties of PBPI in this section, and introduce two weaker variants.

PBVI2 Intuitively, the monotonicity of PBPI would seem to play an important role in its experimental performance, since it guarantees a steadily improving policy values on $b \in B$. It is natural to ask if a less complicated modification to PBVI would suffice. To study this, we modified PBVI to enforce monotonicity of the value function over the sample belief set B. Following the example in Fig. 1(b), it is apparent that the value of some $b \in B$ could decrease if a belief state reachable from b is in the region of reduced value. To guarantee monotonicity, the α-vectors from the previous iteration are preserved if they are dominant for at least at one $b^z_{\pi(b)}, \forall\ b \in B, \forall\ z \in Z$. In effect, this creates a sliding window of α-vectors that covers two iterations of PBVI. This change, summarized in the last two lines of Algorithm 1, is sufficient to ensure monotonicity over the sample belief set B at a computational cost of no more than twice that of original the PBVI. We call this change PBVI2. This change does *not* guarantee monotonically improving policy values, since the value function produced is not necessarily the exact value of any particular policy.

Table 1: Properties of PBPI and its variants.

Method	Policy Evaluation	Policy Improvement	Monotonicity	Policy Size
PBPI	over $b \in \triangle$	over $b \in B$	$V^\pi(b)$ for $b \in B$	increased by $\|B\|$ (worst case)
PBPI2	over $b \in B$	over $b \in B$	$V(b)$ for $b \in B$	at most $\|B\|\|A\|\|Z\|$
PBVI2	N/A	over $b \in B$	$V(b)$ for $b \in B$	at most $\|B\|\|A\|\|Z\|$
PBVI	N/A	over $b \in B$	no guarantee	at most $\|B\|$

PBPI2 Another natural question to ask is whether there is some way to implement a monotonic form of policy iteration without the increase in the policy representation size incurred by PBPI. Toward this end, we introduce PBPI2, which uses PBVI2 for its policy improvement step, and a modified version of PBVI2 for a monotonic, point-based, policy evaluation step. PBPI2 represents the policy explicitly as $\pi(b) = a, \forall\ b \in B$, where B is a finite set of sample beliefs. In the policy evaluation step, the value function of π is estimated by:

$$V^\pi(b) = \sum_{s \in S} R(s, \pi(b)) b(s) + \gamma \sum_{z \in Z} p(z|b, \pi(b)) V^\pi(b^z_{\pi(b)}),$$

$\forall\ b \in B$, which can be solved iteratively in a similar way to PBVI2 without the max operation, producing $|B|$ α-vectors. The policy evaluation step for PBPI2, called *point based policy evaluation* is summarized in Algorithm 3. For policy improvement, PBPI2 performs a single iteration of PBVI2, and stores the action choices from this iteration in a new π. As with PBVI2, PBPI2 cannot guarantee monotonicity in the actual policy values since the policy evaluation step is not exact.

Algorithm 3. Point-based policy evaluation

function $\Gamma' = \text{pbpe}(\pi, \Gamma, B)$
% π is a point-based policy
do forever
  $\Gamma' = \emptyset$
  for each $b \in B$
    $a = \pi(b)$
    $\alpha^z_a = \arg\max_{\alpha \in \Gamma} \alpha \cdot b^z_a$, for every $z \in Z$
    $\alpha'(s) = R(s, a) + \gamma \sum_{z, s'} p(s'|s, a) p(z|s', a) \alpha^z_a(s')$
    if $\alpha' \notin \Gamma'$, **then** $\Gamma' \leftarrow \Gamma' + \alpha'$, **end**
  end
  $\Lambda =$ all $\alpha \in \Gamma$ that are dominant at least at one $b^z_{\pi(b)}$
  $\Gamma' = \Gamma' + \Lambda$
  if $\frac{1}{|B|} \sum_{b \in B} V'(b)$ converges, **then** return Γ', **end**
  $\Gamma = \Gamma'$
end

We summarize the properties of PBPI and its variants in Table 1.

Experimental Results

Scalability of PBPI To illustrate the scalability and the solution quality of PBPI, we test PBPI on four benchmark problems: Tiger-grid, Hallway2, Tag-avoid and the RockSample problems. The first three are among the most widely used benchmarks, and the last one is relatively new and is introduced by Smith & Simmons (2004). This problem can be scaled to an arbitrary size. We test PBPI on domains up to 12,545 states, a limit imposed by available memory in our current Matlab installation.

PBPI is compared to four other state-of-the-art POMDP algorithms: PBVI (Pineau, Gordon, & Thrun 2003), Perseus (Spaan & Vlassis 2005), HSVI (Smith & Simmons 2004; 2005) and BPI (Poupart & Boutilier 2003). In the experiments, the belief set B used in PBPI was generated by setting $\epsilon = 0.6$ and expanding B until it reached the specified size. We terminate PBPI when the change in $\sum_{b \in B} V(b)/|B|$ between two consecutive iterations is below 1% of the change between the initial and the current iteration. Because of the randomness of PBPI (i.e., the selection of belief points for B), we execute PBPI 10 times for each problem using different random seeds, and produce average performance. To test the quality of the learned PBPI policy, we measure the expected reward by the sum of discounted rewards averaged on multiple runs, with the parameter-setting consistent with those used in previous work.

The experimental results are summarized in Table 2. For the four benchmark problems considered, PBPI achieves expected reward competitive with (or higher than) the other algorithms while using a significantly smaller $|B|$ and much less computation time in all cases except some HSVI2 instances. The improvements manifested by PBPI are most marked on large domains, such as Tag-avoid and RockSample. We acknowledge that several aspects of this comparison are unfair in various ways. Our code was implemented in Matlab and run on a new computer, while the results from earlier papers are from older computers running (to the best of our knowledge) C implementations. HSVI2 is known to be a highly optimized C program, while PBPI is implemented fairly straightforward Matlab.

For a fairer comparison, we also implemented the PBVI algorithm ourselves, using the same code used in the policy improvement step of PBPI, and tested on the same belief set and the same convergence criterion used by PBPI. The corresponding solution quality has different levels of degradation compared to that of PBPI. An intuitive explanation of the weaker performance of PBVI is that when planning only on a small belief set B, the size of the PBVI policy is bounded by $|B|$, which may not be enough to express the solution complexity required, while PBPI can add new machine states (equivalently, α-vectors) as needed. In other words, PBVI must rely upon the set of beliefs to encode history information, while PBPI can encode history information in the automaton that it produces.

For PBPI, we include runs with two different size sets of initial belief points, as shown in the rightmost column. The reason for this is to show that PBPI can outperform essentially all competitors in both time and solution quality when using a small initial belief set. When using a larger initial belief set, PBPI takes more time, but produces even better policies.

Table 2: Results on benchmark problems. Results marked ($*$) were computed by us using Matlab (without C subroutines) run on a 3.4GHz Pentium 4 machine.

Method	Reward	Time (s)	$\|\Gamma\|$	$\|B\|$
Tiger-grid $\|S\| = 36, \|A\| = 5, \|Z\| = 17$				
PBVI (Pineau *et al.* 2003)	2.25	3448	n.v.	470
Perseus (Spaan & Vlassis 2005)	2.34	104	134	1000
HSVI1 (Smith & Simmons 2004)	2.35	10341	4860	n.v.
HSVI2 (Smith & Simmons 2005)	2.30	52	1003	n.v.
BPI (Poupart 2005)	2.22	1000	120	n.a.
PBVI ($*$)	2.05	7	130	135
PBPI ($*$)	2.08	14	1739	90
PBPI ($*$)	2.24	51	3101	135
Hallway2 $\|S\| = 92, \|A\| = 5, \|Z\| = 17$				
PBVI (Pineau *et al.* 2003)	0.34	360	n.v.	95
Perseus (Spaan & Vlassis 2005)	0.35	10	56	1000
HSVI1 (Smith & Simmons 2004)	0.35	10010	1571	n.v.
HSVI2 (Smith & Simmons 2005)	0.35	1.5	114	n.v.
BPI (Poupart 2005)	0.32	790	60	n.a.
PBVI ($*$)	0.33	1.9	20	20
PBPI ($*$)	0.34	1.8	171	10
PBPI ($*$)	0.35	3.1	320	20
Tag-avoid $\|S\| = 870, \|A\| = 5, \|Z\| = 30$				
PBVI (Pineau *et al.* 2003)	-9.18	180880	n.v.	1334
Perseus (Spaan & Vlassis 2005)	-6.17	1670	280	10000
HSVI1 (Smith & Simmons 2004)	-6.37	10113	1657	n.v.
HSVI2 (Smith & Simmons 2005)	-6.36	24	415	n.v.
BPI (Poupart 2005)	-6.65	250	17	n.a.
PBVI ($*$)	-12.59	724	198	300
PBPI ($*$)	-6.54	365	485	100
PBPI ($*$)	-5.87	1133	818	300
RockSample[5,7] $\|S\| = 3201, \|A\| = 12, \|Z\| = 2$				
HSVI1 (Smith & Simmons 2004)	23.1	10263	287	n.v.
PBVI ($*$)	18.1	8494	539	800
PBPI ($*$)	24.1	2835	1045	400
PBPI ($*$)	24.5	8743	1858	800
RockSample[7,8] $\|S\| = 12545, \|A\| = 13, \|Z\| = 2$				
HSVI1 (Smith & Simmons 2004)	15.1	10266	94	n.v.
HSVI2 (Smith & Simmons 2005)	20.6	1003	2491	n.v.
PBVI ($*$)	12.9	43706	351	500
PBPI ($*$)	20.8	11233	585	200
PBPI ($*$)	21.2	29448	1257	500

n.a. = not applicable n.v. = not available

Robustness and Performance Profiles of PBPI To examine the robustness and scaling of the PBPI algorithm, we compare the performance of PBPI against PBVI with increasing $|B|$. Again, the belief set B was generated by setting $\epsilon = 0.6$ and expanding B until it reached the specified size. We execute the PBPI algorithm 10 times, and produce average performance. The experimental results on RockSample[7,8] are provided in Fig. 2. Figure 2(a) shows the mean and the range of the expected rewards over 10 random runs. PBPI typically requires much fewer belief points than PBVI for comparable solution quality, and the solution of PBPI is more robust, as the variances on the PBPI results are significantly smaller than that of PBVI. Although each iteration of PBPI is more expensive than that of PBVI, PBPI typically requires much fewer iterations to converge. Thus, the final computation time for PBPI and PBVI are comparable on a given belief set. Considering PBPI needs much fewer belief points than PBVI for comparable solution quality, PBPI is more efficient than PBVI per unit time, as demonstrated in Fig. 2(b). Space does not permit reporting this performance profile for each of the benchmark problems, but the results reported in this section are representative. The two PBPI entries in Table 2 are intended to represent two points on the performance profile for PBPI for each of the sample problems.

Variants of PBPI In the last experiment, we compared the performance of PBPI, PBPI2 and PBVI2 with an increasing number of belief points $|B|$ on the Tag-avoid problem, with the results shown in Fig. 3 averaged on 10 random realizations. This experiment is designed to study the impact of monotonicity and policy evaluation on the performance of POMDP algorithms. Figure 3(a) demonstrates the superior solution qualities of PBPI and its variants over that of PBVI. This may be explained by the monotonicity of different algorithms. PBPI guarantees improved *policies*, while PBPI2 and PBVI2 ensure only monotonic approximate value functions, and there is no guarantee at all for PBVI. Figure 3(b) shows the computation time of the different algorithms with increasing $|B|$. In this case, PBPI2 has the smallest computation time. This is consistent with the conventional observation that policy iteration converges faster than value iteration. Further, because PBPI2 uses a less expensive, approximate policy evaluation step and a smaller policy representation, PBPI2 is faster than PBPI for a given $|B|$. A more fair comparison is given in Fig. 3(c), in which the solution quality is compared along with increasing computation time. In this case, PBPI is the most efficient algorithm considered, per unit time. These results are typical of results on the other benchmark problems.

Related Work

Since the appearance of Hansen's policy iteration, several algorithms have been proposed along the line of policy iteration for searching in the policy space represented by finite-state controllers. All the algorithms are approximate policy iteration in order to alleviate the computation load of the exact policy iteration. In particular, Hansen's heuristic search policy iteration (Hansen 1998) replaces the exact policy improvement with a heuristic search algorithm and updates the FSC as the exact policy iteration does; Meuleau *et al.* (1999) use gradient ascent (GA) to directly search for a stochastic policy represented by an FSC of bounded size, even though this approach is prone to be trapped in local optima; more recently, Poupart *et al.* (2003) proposed a bounded policy iteration (BPI) algorithm, which finds a stochastic policy represented by an FSC of bounded size (via linear programming) and avoids obvious local optima by adding one machine state at each iteration. Our PBPI algorithm is more related to Hansen's heuristic search policy iteration in the

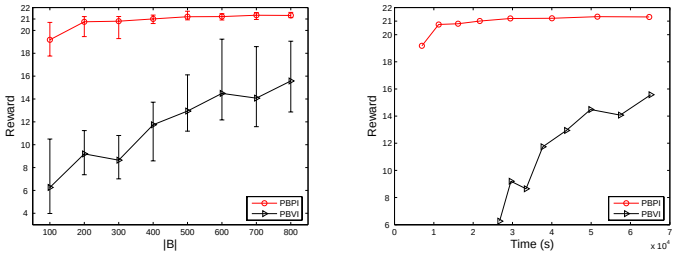

Figure 2: Performances of PBPI and PBVI on RockSample[7,8] along with (a) an increasing $|B|$, and (b) computation time.

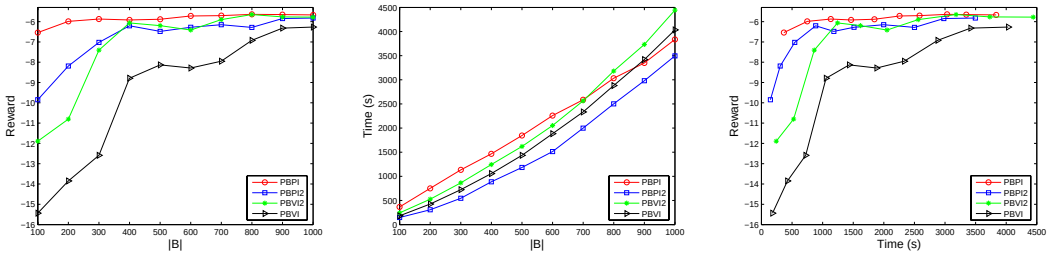

Figure 3: Performances of PBPI and its variants on Tag-avoid, with the comparison made in terms of (a) solution quality, (b) computation time, and (c) solution quality in a given time.

sense that both methods focus on deterministic FSCs and replace the expensive exact policy improvement step with approximate algorithms.

Conclusions

We have proposed a point-based policy iteration (PBPI) algorithm for infinite-horizon POMDPs. PBPI integrates the fast convergence of policy iteration and the high efficiency of PBVI for policy improvement, guaranteeing improved policies at each iteration. Experiments on several benchmark problems demonstrate the scalability and robustness of the proposed PBPI algorithm. One possible area for future work would be to use the HSVI (Smith & Simmons 2005) heuristic to choose belief points for PBPI.

Acknowledgments

The authors wish to thank the anonymous reviewers for their constructive suggestions, and M. Spaan and N. Vlassis for sharing their Matlab POMDP file parser online. This work was supported in part by the Sloan foundation, and by NSF IIS award 0209088. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

Cassandra, A. R.; Littman, M.; and Zhang, N. 1997. Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In *UAI 13*, 54–61.

Hansen, E. A. 1998. Solving POMDPs by searching in policy space. In *UAI 14*, 211–219.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101:99–134.

Meuleau, N.; Kim, K. E.; Kaelbling, L. P.; and Cassandra, A. R. 1999. Solving POMDPs by searching the space of finite policies. In *UAI 15*, 417–426.

Pineau, J.; Gordon, G.; and Thrun, S. 2003. Point-based value iteration: An anytime algorithm for POMDPs. In *IJCAI*, 1025–1032.

Poupart, P., and Boutilier, C. 2003. Bounded finite state controllers. In *NIPS 16*.

Poupart, P. 2005. *Exploiting structure to efficiently solve large scale partially observable Markov decision processes*. Ph.D. Dissertation, University of Toronto, Toronto.

Puterman, M. L., and Shin, M. C. 1978. Modified policy iteration algorithms for discounted Markov decision problems. *Management Science* 24(11).

Smallwood, R. D., and Sondik, E. J. 1973. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research* 21(5):1071–1088.

Smith, T., and Simmons, R. 2004. Heuristic search value iteration for POMDPs. In *UAI 20*.

Smith, T., and Simmons, R. 2005. Point-based POMDP algorithms: Improved analysis and implementation. In *UAI 21*.

Sondik, E. J. 1978. The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Operations Research* 26(2):282–304.

Spaan, M., and Vlassis, N. 2005. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research* 24:195–220.