Prioritizing Point-Based POMDP Solvers*

Guy Shani, Ronen I. Brafman, and Solomon E. Shimony

Department of Computer Science, Ben-Gurion University, Beer-Sheva, Israel

Abstract. Recent scaling up of POMDP solvers towards realistic applications is largely due to point-based methods such as PBVI, Perseus, and HSVI, which quickly converge to an approximate solution for medium-sized problems. These algorithms improve a value function by using *backup* operations over a single belief point. In the simpler domain of MDP solvers, prioritizing the order of equivalent backup operations on states is well known to speed up convergence.
We generalize the notion of prioritized backups to the POMDP framework, and show that the ordering of backup operations on belief points is important. We also present a new algorithm, Prioritized Value Iteration (PVI), and show empirically that it outperforms current point-based algorithms. Finally, a new empirical evaluation measure, based on the number of backups and the number of belief points, is proposed, in order to provide more accurate benchmark comparisons.

1 Introduction

Many interesting reinforcement learning (RL) problems can be modeled as partially observable Markov decision problems (POMDPs), yet POMDPs are frequently avoided due to the difficulty of computing an optimal policy. Research has focused on approximate methods for computing a policy (see e.g. [8],[7]). A standard way to define a policy is through a value function that assigns a value to each belief state, thereby also defining a policy over the same belief space. Sondik [9] show that this value function can be represented by a set of vectors and is therefore piecewise linear and convex.

A promising approach for computing value functions is the point-based method, where a value function is computed over a finite set of reachable belief points, in the hope that it would generalize well to the entire belief space. Generalization is possible through the use of the vector representation of a value function.

Improving a value function represented by vectors can be done by performing a *backup* operation over a single belief state, resulting in a new vector that can be added to the value function. Even though a vector is computed for a single belief state, it defines a value over the entire belief space, though this value may not be optimal for many belief states. A single backup operation can therefore, and in many cases does, improve the value function for numerous belief points. Backup operations are relatively expensive, and POMDP approximation algorithms can be improved by reducing the number of backup operations needed to approximate the optimal policy.

* Partially supported by the Lynn and William Frankel Center for Computer Sciences, and by the Paul Ivanier Center for Robotics and Production Management at BGU. Ronen Brafman is partially supported by NSF grants SES-0527650 and IIS-0534662.

For the simpler domain of Markov decision processes (MDPs), it was previously observed (e.g. [13]) that the order by which states are updated can change the convergence rate of the value function. For example, as the value for a state is influenced by the values of its successors it is more useful to execute a backup operation for a state only after values for its successors are computed. In an MDP it is also easy to find the set of predecessors for a given state making backward state space traversals possible. Such methods can be viewed as ordering backups by decreasing state *priorities*.

This paper aims at taking advantage of prioritized backups in a similar manner for POMDPs. Since a direct implementation of the techniques used for MDPs is not possible, this issue is nontrivial. First, one cannot efficiently find the set of predecessors for a belief state, which may have unbounded size. Second, a backup operation for a belief state potentially improves the value for many other belief states as well, and therefore affecting belief states that are not direct predecessors of the improved state.

Point-based methods tackle these problems by using only a finite subset of the belief space, reachable from the initial belief state. The main contribution of this paper is in showing how priorities can be computed for this finite set of belief points, and clearly demonstrating the resulting improvement in speed of convergence towards an approximate policy. We can hence provide prioritized versions for the PBVI [7] and Perseus [12] algorithms, as well as a new prioritized algorithm, Prioritized Value Iteration (PVI), which outperforms the unprioritized algorithms.

Another, methodological contribution of this paper is related to the schemes used for reporting experimental results evaluating the performance of point-based algorithms. Previous researchers have implemented their own algorithms and compared the results to previous published results, usually reporting Average Discounted Reward (ADR) as a measure of the quality of the computed policy, and convergence time, over well-known benchmarks. Observe that while the ADR of a policy is identical over different implementations, the convergence time is an insufficient measurement. Execution time for an algorithm is highly sensitive to variations in machines (CPU speed, memory capacity), selected platform (OS, programming language) and implementation efficiency. We comply with the commonly used result reporting scheme, but additional measures are also reported, which may provide meaningful comparisons in future publications.

2 Background and Related Work

2.1 MDPs, POMDPs and the belief-space MDP

A Markov Decision Process (MDP) is a tuple $\langle S, A, tr, R\rangle$ where S is a set of world states, A is a set of actions, $tr(s, a, s')$ is the probability of transitioning from state s to state s' using action a, and $R(s, a)$ defines the reward for executing action a in state s. An MDP models an agent that can directly observe its state in the environment.

A Partially Observable Markov Decision Process (POMDP) is a tuple $\langle S, A, tr, R, \Omega, O, b_0\rangle$ where S, A, tr, R are the same as in an MDP, Ω is a set of observations and $O(a, s, o)$ is the probability of observing o after executing a and reaching state s. A POMDP is a better model for real agents, such as robots, that do not have direct access to the current state of world but rather observe the world through a set of

sensors that provide noisy observations. The agent hence must maintain a *belief* over its current state — a vector b of probabilities such that $b(s)$ is the probability that the agent is at state s. Such a vector is known as a belief state or *belief point*. b_0 defines the initial belief state before the agent has executed an action or received an observation.

Given a POMDP it is possible to define the belief-space MDP — an MDP over the belief states of the POMDP. The transition from belief state b to belief state b' using action a is deterministic given an observation o and defines the τ transition function. That is, we denote $b' = \tau(b, a, o)$ where:

$$b'(s') = \frac{O(a, s', o) \sum_s b(s) tr(s, a, s')}{pr(o|b, a)} \tag{1}$$

$$pr(o|b, a) = \sum_s b(s) \sum_{s'} tr(s, a, s') O(a, s', o) \tag{2}$$

Thus, τ can be computed in time $O(|S|^2)$.

2.2 Value Functions for POMDPs

It is well known that the value function V for the belief-space MDP can be represented as a finite collection of $|S|$-dimensional vectors known as α vectors. Thus, V is both piecewise linear and convex [9]. A policy over the belief space is defined by associating an action a to each vector α, so that $\alpha \cdot b = \sum_s \alpha(s) b(s)$ represents the value of taking a in belief state b and following the policy afterwards. It is therefore standard practice to compute a value function — a set V of α vectors. The policy π_V is immediately derivable using:

$$\pi_V(b) = \text{argmax}_{a:\alpha_a \in V} \, \alpha_a \cdot b \tag{3}$$

The belief-space value function can be iteratively computed

$$V_{n+1}(b) = \max_a [b \cdot r_a + \gamma \sum_o pr(o|a, b) V_n(\tau(b, a, o))] \tag{4}$$

where $r_a(s) = R(s, a)$ is a vector representation of the reward function. The computation of the next value function $V_{n+1}(b)$ out of the current V_n (Equation 4) is known as a *backup* step, and can be efficiently implemented [2, 7] by:

$$g^{\alpha}_{a,o}(s) = \sum_{s'} O(a, s', o) tr(s, a, s') \alpha^i(s') \tag{5}$$

$$g^b_a = r_a + \gamma \sum_o \text{argmax}_{g^{\alpha}_{a,o}:\alpha \in V} \, b \cdot g^{\alpha}_{a,o} \tag{6}$$

$$backup(b) = \text{argmax}_{g^b_a : a \in A} \, b \cdot g^b_a \tag{7}$$

Note that the $g^{\alpha}_{a,o}$ computation (Equation 5) does not depend on b and can therefore be cached for future backups. All the algorithms we implemented use caching to speed up backup operations. Without caching the execution time of the backup operation takes $O(|S|^2|V||\Omega||A|)$. In most benchmark POMDPs considered in our experiments, the

number of actions and size of observation space are bounded, leading to a complexity of $O(|S|^2|V|)$ per backup step.

While it is possible to update V over the entire belief space, hence computing an optimal policy [2], the operation is computationally hard. Various approximation schemes attempt to decrease the complexity of computation, potentially at the cost of optimality.

The vector representation is suitable only for lower bounds over the optimal value function. When a value function is given using some other representation, such as a direct mapping between belief states and values, one can define the H operator, known as the Bellman update, that computes a value function update as:

$$Q_V(b,a) = b \cdot r_a + \gamma \sum_o pr(o|a,b) V_n(\tau(b,a,o)) \quad (8)$$

$$HV(b) = \max_a Q_V(b,a) \quad (9)$$

The computation time of the H operator is $O(T_v|S|^2|O||A|)$, where T_v is the computation time of a specific belief point value using the value function V.

2.3 Point Based Value Iteration

Computing an optimal value function over the entire belief space does not seem to be a feasible approach. A possible approximation is to compute an optimal value function over a subset of the belief space [5]. Note that an optimal value function for a subset of the belief space is no more than an approximation of a full solution. We hope, however, that the computed value function will generalize well for unobserved belief states.

Point-based algorithms [7, 12, 11] choose a subset of the belief points that is reachable from the initial belief state through different methods, and compute a value function only over these belief points.

Algorithm 1 PBVI

Function PBVI
1: $B \leftarrow \{b_0\}$
2: **while** true **do**
3: $Improve(V, B)$
4: $B \leftarrow Expand(B)$

Function Improve(V,B)
Input: V — a value function
Input: B — a set of belief points
1: **repeat**
2: **for each** $b \in B$ **do**
3: $\alpha \leftarrow backup(b)$
4: $add(V, \alpha)$
5: **until** V has converged

Function Expand(B)
Input: B — a set of belief points
1: $B' \leftarrow B$
2: **for each** $b \in B$ **do**
3: $Succ(b) \leftarrow \{b' | \exists a, \exists o\, b' = \tau(b,a,o)\}$
4: $B' \leftarrow B' \cup argmax_{b' \in Succ(b)} dist(B, b')$
5: **return** B'

Point Based Value Iteration (PBVI) [7] (Algorithm 1), begins with b_0, and at each iteration computes an optimal value function for the current belief points set. After

convergence the belief points set is expanded by adding the most distant immediate successors of the previous set. Following Pineua et al. we used the L_2 distance metric.

Spaan and Vlassis [12] explore the world randomly, gathering a set B of belief points, and then executing the Perseus[1] algorithm (Algorithm 2). Perseus appears to provide good approximations with small sized value functions rapidly. However, it is very stochastic due to the random selection of belief points and the random selection of backup operations. These random selections cause a high variation in performance and in more complicated problems may cause the algorithm to fail to converge at all.

Algorithm 2 Perseus

Input: B — a set of belief points

repeat
  $\tilde{B} \leftarrow B$
  while $\tilde{B} \neq \phi$ **do**
    Choose $b \in \tilde{B}$
    $\alpha \leftarrow backup(b)$
    if $\alpha \cdot b \geq V(b)$ **then**
      $add(V, \alpha)$
      $\tilde{B} \leftarrow \{b \in \tilde{B} : \alpha \cdot b < V(b)\}$
until V has converged

Smith and Simmons [10, 11] present the Heuristic Search Value Iteration algorithm (HSVI - Algorithm 3) that maintains both an upper bound $\bar{V}$ and lower bound $\underline{V}$ over the value function. HSVI traverses the belief space following the $\bar{V}$ policy, greedily selecting successor belief points where the gap between the bounds is the largest, until some stopping criteria has been reached. Afterwards it executes backup and H operator updates over the observed belief points on the explored path in a reversed order. The policy computed by HSVI is based on $\underline{V}$. $\bar{V}$ is used only for exploration.

Algorithm 3 HSVI

Function HSVI
Initialize $\underline{V}$ and $\bar{V}$
while $\underline{V}(b_0) - \bar{V}(b_0) > \epsilon$ **do**
  $Explore(b_0, 0)$

Function Explore(b, d**)**
Input: b — a belief state
Input: d — depth of recursion
if $\underline{V}(b) - \bar{V}(b) < \epsilon\gamma^{-d}$ **then**
  $a^* \leftarrow \text{argmax}_a\, Q_{\bar{V}}(b, a')$ (Equation 8)
  $o^* \leftarrow \text{argmax}_o(\bar{V}(\tau(b, a, o)) - \underline{V}(\tau(b, a, o)))$
  Explore$(\tau(b, a^*, o^*), d + 1)$
  $add(\underline{V}, backup(b, \underline{V}))$
  $\bar{V} \leftarrow HV(b)$

[1] We present here a single value function version of Perseus.

3 Prioritized Point Based Value Iteration

Point based algorithms compute a value function using α vectors by iterating over some finite set of belief points and executing a sequence of backup operations over these belief points. For each set of belief points there are many possible sequences of backup executions. As our goal is to approximate the value function as quickly as possible, we say that a backup sequence seq_1 is better than sequence seq_2 if seq_1 is shorter than seq_2, and produces a policy which is no worse than the one produced by seq_2.

We suggest creating (hopefully) better sequences using a heuristic that predicts useful backups. The heuristic computation must be efficient so that the overhead of computing the heuristic does not outweigh any savings achieved by performing fewer backups.

3.1 Prioritizing MDP Solvers

A comparable scheme used for prioritizing in MDP solvers, suggests performing the next backup on the MDP state that maximizes the Bellman error:

$$e(s) = max_a[R(s,a) + \sum_{s'} tr(s,a,s')V(s')] - V(s). \tag{10}$$

$e(s)$ measures the change in $V(s)$ from performing a backup. Wingate and Seppi[13] present a very simple version of prioritized value iteration for MDPs (Algorithm 4).

Algorithm 4 Prioritized Value Iteration for MDPs

1: $\forall s \in S, V(s) \leftarrow 0$
2: **while** V has not converged **do**
3: $\quad s \leftarrow \text{argmax}_{s' \in S}\, e(s')$
4: $\quad backup(s')$

A key observation for the efficiency of their algorithm is that after a backup operation for state s, the Bellman error recomputation need be performed only for the predecessors of s $\{s' : \exists a, tr(s',a,s) \neq 0\}$.

3.2 Prioritizing POMDP Solvers

While the Bellman error generalizes well to POMDPs:

$$e(b) = max_a[r_a \cdot b + \sum_o pr(o|a,b)V(\tau(b,a,o))] - V(b) \tag{11}$$

there are two key differences between applying priorities to MDPs and POMDPs; First, a backup update affects more than a single state. A new vector usually improves the local neighborhood of its witness belief point. Second, the set of predecessors of a belief state cannot be efficiently computed, and its size is potentially unbounded.

Moreover, even supposing that some similarity metric for finding the neighborhood of a belief point were defined, and that computation of the predecessor set were only for the finite set of belief points we use, directly applying the approach would still be infeasible. In practice, algorithms such as Perseus, frequently converge to an optimal solution while computing fewer backups than the number of belief points in the finite set. Pre-computations such as similarity matrixes will take more time than the original algorithm they are designed to improve in the first place.

As we cannot find the set of belief states affected by the backup operation directly, we recompute the Bellman error for all belief states after every backup from scratch. When the number of belief points we use is relatively small this computation can be done without seriously damaging the performance. As the size of the problem — states, actions, observations and belief set size — increases, we can no longer afford the overhead of recomputing the Bellman error for all belief states.

We take a stochastic approach, sampling (uniformly, with repetitions) a subset of the belief points set and computing the Bellman error only for the sampled subset. If the subset does not contain a point with positive error, we sample again from the remaining subset until a belief point with positive error is found. If there is no belief point with positive Bellman error then the value function reached a fixed point and therefore converged to an optimal solution over the finite set of belief points, and cannot be improved using point-based backups.

3.3 Prioritizing Existing Algorithms

Prioritizing Perseus is straight forward. The choose step is implemented in Perseus as a uniform selection among any of the current belief points inside $\tilde{B}$. Prioritized Perseus uses $e(b)$ (Equation 11) to choose a belief point whose value can be improved the most.

PBVI improves its value function (Algorithm 1, line 3) by arbitrarily preforming backups overs belief points. We replace this inefficient computation of the Improve operation with our PVI algorithm (see Section 3.4). As the number of points used by PBVI is relatively small, no sampling was used when computing the Bellman error.

3.4 Prioritized Value Iteration

Finally, we present an independent algorithm — Prioritized Value Iteration (PVI). Like Perseus, PVI computes a value function over a pre-collected fixed set of belief points. However, Perseus operates in iterations over the set of belief points, attempting to improve all belief points between considering the same point twice. PVI considers at each step every possible belief state for improvement. It is likely, therefore, that some belief states will be backed up many times, while other belief states will never be used.

Algorithm 5 presents our PVI algorithm. Note however that while the algorithm described here is the clean version of PVI, in practice we implement the $argmax$ operation (line 2) using our sampling technique described above. If the prioritization metric is good, PVI executes a shorter sequence of backup operations. Indeed, experiments show that it uses significantly fewer backup operations than Perseus using our locally greedy Bellman error prioritization metric.

Algorithm 5 Prioritized Value Iteration

Input: B — a set of belief points
1: **while** V has not converged **do**
2: $b^* \leftarrow \text{argmax}_{b \in B}\, e(b)$
3: $\alpha \leftarrow backup(b^*)$
4: $add(V, \alpha)$

4 Empirical Evaluations

4.1 Improved Evaluation Metrics

Previous researchers [1, 7, 11, 12, 6] limit their reported results to execution time, ADR and in some cases the number of vectors in the final value function.

Value function evaluation — Average discounted reward (ADR) is computed by simulating the agent interaction with the environment over a number of steps (called a *trial*) and averaging over a number of different trials: $\frac{\sum_{i=0}^{\#trials} \sum_{j=0}^{\#steps} \gamma^j r_j}{\#trials}$.

ADR is widely agreed as a good evaluation of the value of a value function. It can, however, present very noisy results when the number of trials or the number of steps is too small. To remove such noise we used a first order filter with weight 0.5. We stopped the execution once the filtered ADR has converged to a predefined target.

Execution time — As all algorithms discussed in this paper compute a value function using identical operations such as backups, τ function computations, and dot products ($\alpha \cdot b$), it seems that recording the number of executions of those basic building blocks of the algorithm is more informative than just reporting CPU time.

Memory — The size of the computed value function and the total amount of maintained belief points are good estimates for the memory capacity required for the computation of the algorithm.

4.2 Experimental Setup

In order to test our prioritized approach, we tested all algorithms on a number of benchmarks from the point-based literature: Hallway, Hallway2 [4] (two maze navigation problems), TagAvoid [7] (a robot must capture another escaping robot) and RockSample [10] (a robot identifies and visits valuable rocks on a map). Table 2 contains the problem measurements for the benchmarks including the size of the state space, action space and observation space, the number of belief points in the set $|B|$ used for Perseus, Prioritized Perseus and PVI, and the ADR measurement error over $10,000$ trials.

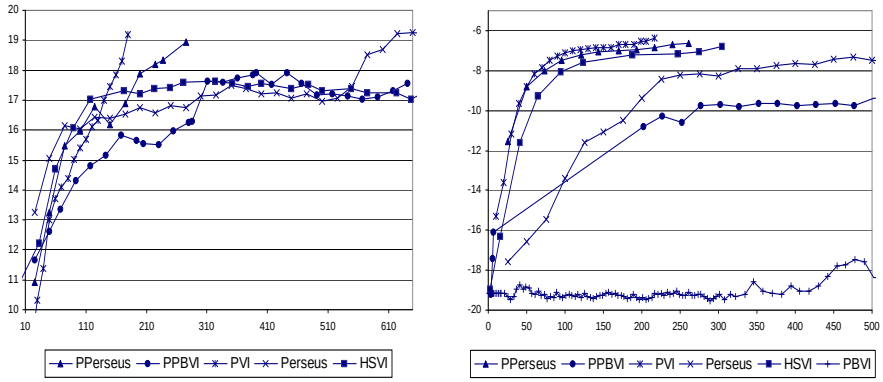
We implemented in Java a standard framework that incorporated all the basic operators used by all algorithms such as vector dot products, backup operations, τ function and so forth. All reported results were gathered by executing the algorithms on identical machines — x86 64-bit machines, dual-proc, processor speed 2.6Ghz, 4Gb memory, 2Mb cache, running linux and JRE 1.5.

As previous researchers have already shown the maximal ADR achievable by their methods, we focus our attention on convergence speed of the value function to the reported ADR. We executed all algorithms, interrupting them from time to time in order to

compute the efficiency of the current value function using ADR over 5000 trials. Once the filtered ADR has reached the same level as reported in past publications execution was stopped. The reported ADR was then measured over additional $10,000$ trials (error in measurement is reported in Table 2).

We pre-computed 5 different sets of belief points for each problem, by simulating an interaction with the system following the Q_{MDP} policy with an ϵ-greedy exploration factor ($\epsilon = 0.1$). These were then used for algorithms that require a given set of belief states B — Perseus, Prioritized Perseus and PVI. For each such belief points set we ran 5 different executions with different random seeds resulting in 25 different runs for each stochastic method. The belief points set size for each problem is specified in Table 2. Using Q_{MDP} for gathering B allowed us to use a smaller belief set than [12].

Algorithms that are deterministic by nature — PBVI, Prioritized PBVI and HSVI — were executed once per problem.



(a) (b)

Fig. 1. Convergence on the Rock Sample 5,5 problem (a) and the Tag Avoid problem (b). The X axis shows the number of backups and the Y axis shows ADR.

4.3 Results

Table 1 presents our experimental results. For each problem and method we report: the resulting ADR, the size of the final value function ($|V|$), the CPU time until convergence, the number of backups, of $g^{\alpha}_{a,o}$ operations, of computed belief states, of τ function computations, and of dot product operations.

The reported numbers do not include the repeated expensive computation of the ADR, or the initialization time (identical for all algorithms). Results for algorithms that require a pre-computed belief space do not include the effort needed for this pre-computation. We note, however, that it took only a few seconds (less than 3) to compute the belief space over all problems.

Method	**ADR**	$\|V\|$	**Time (secs)**	**# Backups**	**#** $g^{\alpha}_{a,o}$ x 10^6	**# belief states** x 10^4	**#** τ x 10^3	**#** $\alpha \cdot b$ x 10^6
Hallway								
PVI	0.517±0.0027	144±32	**75**±32	**504**±107	3.87±1.75	1.99±0.04	4.8±9.8	13.11±5.19
PPerseus	0.517±0.0025	173±43	126±47	607±166	5.52±2.95	1.99±0.04	4.8±9.8	26.87±9.2
Perseus	0.517±0.0024	466±164	125±110	1456±388	31.56±27.03	0.03±0	0±0	32.07±27.16
PPBVI	0.519	235	95	725	9.09	1.49	13.45	25.72
PBVI	0.517	253	118	3959	31.49	1.49	15.79	31.69
HSVI	0.516	182	314	634	5.85	3.4	34.52	6.67
Hallway2								
PVI	0.344±0.0037	234±32	75±20	262±43	2.59±0.84	2.49±0.11	5.47±9.99	6.96±2.01
Pperseus	0.346±0.0036	273±116	219±155	343±173	4.76±5.48	2.49±0.11	5.25±9.99	18.97±12.79
Perseus	0.344±0.0034	578±95	134±48	703±120	17.03±6.08	0.03±0	0±0	17.31±6.13
PPBVI	0.347	109	**59**	**137**	0.61	2.03	10.77	4.22
PBVI	0.345	128	76	1279	7.96	1.52	5.59	8.02
HSVI	0.341	172	99	217	1.56	2.11	11.07	1.81
Tag Avoid								
PVI	-6.467±0.19	204±38	**40**±12	211±38	0.42±0.19	0.16±0	0.5±1.02	0.95±0.25
PPerseus	-6.387±0.18	260±43	105±26	265±44	5.27±1.8	1.73±0.02	5.68±11.59	12.82±3.01
Perseus	-6.525±0.20	365±69	212±174	11242±10433	28.69±32.09	0.04±0	0±0	30.78±33.96
PPBVI	-6.271	167	50	**168**	2.09	0.41	32.11	2.45
PBVI	-6.6	179	1075	21708	407.04	0.41	56.53	409.5
HSVI	-6.418	100	52	304	0.5	0.29	1.74	0.53
Rock Sample 4,4								
PVI	17.725±0.32	231±41	4±2	232±42	0.36±0.14	0.41±0.01	1.17±2.38	1.74±0.42
PPerseus	17.574±0.35	229±26	5±2	228±27	0.34±0.08	0.41±0.01	1.17±2.38	1.91±0.29
Perseus	16.843±0.18	193±24	158±33	24772±5133	59.96±13.06	0.05±0	0±0	66.52±14.25
PPBVI	18.036	256	229	265	0.62	2.43	55.31	9.46
PBVI	18.036	179	442	52190	113.16	1.24	35.47	119.8
HSVI	18.036	123	**4**	**207**	1.08	0.1	1.17	1.09
Rock Sample 5,5								
PVI	19.238±0.07	357±56	**21**±7	**362**±63	0.99±0.36	0.46±0.01	1.37±2.79	3.39±0.87
PPerseus	19.151±0.33	340±53	20±6	339±53	0.88±0.28	0.46±0.01	1.37±2.79	3.5±0.73
Perseus	19.08±0.36	413±56	228±252	10333±9777	60.34±66.62	0.05±0	0±0	63.17±69.21
PPBVI*	17.97	694	233	710	4.95	0.95	17.97	18.12
PBVI*	17.985	353	427	20616	72.01	0.49	11.28	75.64
HSVI	18.74	348	85	2309	10.39	0.26	2.34	10.5
Rock Sample 5,7								
PVI	22.945±0.41	358±88	**89**±34	359±89	1.28±0.64	0.29±0.01	0.73±1.49	2.98±1.16
Pperseus	22.937±0.70	408±77	118±37	407±77	1.61±0.59	0.29±0.01	0.73±1.49	4.09±0.98
Perseus	23.014±0.77	462±70	116±31	1002±195	5.18±1.9	0.02±0	0±0	5.36±1.93
PPBVI*	21.758	255	117	**254**	0.61	0.23	2.71	1.59
PBVI*	22.038	99	167	2620	3.05	0.15	1.66	3.23
HSVI	23.245	207	156	314	0.83	0.71	4.2	0.88

Table 1. Performance measurements. The algorithms that executed fewer backups and converged faster are bolded.

To illustrate the convergence of the algorithms we have also plotted the convergence of the ADR vs. the number of backups an algorithm preforms in Figure 1. The graphs contain data collected over separate executions with fewer trials (500 instead of 10000) so Table 2 is more accurate.

HSVI is the only method that also maintains an upper bound over the value function ($\bar{V}$). Table 3 contains additional measurements for the computation of the upper bound: the number of points in $\bar{V}$, the number of projections of other points onto the upper bound, and the number of upper bound updates ($HV(b)$ — Equation 9).

PBVI and PPBVI failed in two cases (Rock Sample 5,5 and 5,7 — marked with an asterix) to improve the reported ADR even when allowed more time to converge.

Problem	$\|S\|$	$\|A\|$	$\|O\|$	$\|B\|$	ADR Error
Hallway	61	5	21	250	± 0.0015
Hallway2	93	5	17	300	± 0.004
Tag Avoid	870	5	30	350	± 0.045
Rock Sample 4,4	257	9	2	500	± 0.075
Rock Sample 5,5	801	10	2	500	± 0.3
Rock Sample 5,7	3201	12	2	500	± 0.25

Table 2. Benchmark problem parameters

Problem	$\|\bar{V}\|$	**#$\bar{V}$(b)**	**#HV(b)**	$\|B\|$
Hallway	423	106132	1268	523
Hallway2	232	37200	434	171
Tag Avoid	1101	29316	1635	248
Rock Sample 4,4	344	6065	414	176
Rock Sample 5,5	801	101093	6385	1883
Rock Sample 5,7	3426	9532	628	268

Table 3. Upper bound measurements for HSVI

4.4 Discussion

Our results clearly show that an informed choice of the order by which backups are preformed over a predefined set of points improves the convergence speed. The most significant result is that our new algorithm, PVI, is among the quickest to converge in all but one test-bed (in Hallway2 it is outperformed only by PPBVI). The efficiency of PVI's backup choices shows up nicely in Figure 1, where we see the steep improvement of PVI. Its improvement seems to be the steepest among the algorithms tested, indicating that it is a good choice for a fast approximation algorithm.

We can also see that, in general, prioritization helps each specific algorithm. We see it clearly in the case of PBVI – its running time is always faster with prioritization – whereas for Perseus there is one domain (Hallway2) in which the prioritized version is significantly slower, and two domains where the performance is virtually the same (Hallway and Rock Sample 5,7).

We can see an even more pronounced effect of prioritization on the number of backups, both between the two version of PBVI and Perseus, and with respect to PVI. In all these cases, there is an order of magnitude reduction in the number of backup operations when the next backup to preform is chosen in an informed manner. However, we also see that there is a penalty we pay for computing the Bellman error, so that the saving in backups does not fully manifest in execution time. Nevertheless, this investment is well worth it, as the overall performance improvement is clear. Note that the ADR to which the different algorithms converge is not identical, but the differences are minor, never exceeding 2%, making all ADRs shown equivalent, for all practical purposes.

In many case, HSVI executes less backups than other algorithms. Indeed, one may consider HSVI's selection of belief space trajectories as a form of prioritization metric. As such, we note that in most cases our form of backup selection exhibits superior runtime to HSVI, even when the number of backups HSVI uses is smaller. This is due to the costly maintenance of the value function upper bound.

5 Conclusions

This paper demonstrates how point-based POMDP solvers such as PBVI and Perseus can greatly benefit from intelligent selection of the order of backup operations, and that such selection can be performed efficiently, so that the algorithms' overall performance improves. It provides an extensive experimental analysis of different aspects

of the performance of current point-based algorithms as well as their prioritized versions on popular domains from the literature. It also presents an independent algorithm — Prioritized Value Iteration (PVI) — that outperforms current point-based algorithm on a large set of benchmarks converging faster toward comparable values of ADR. Given that point-based algorithms are the methods of choice for approximately solving POMDPs, PVI appears to be the fastest current approximation algorithm for POMDPs.

All the prioritized algorithms described in this paper use the same heuristic measure, the Bellman error, to decide on the sequence of backups. The method for selecting the order of backups using the Bellman error is pointwise greedy. While this choice may be optimal in the context of MDPs, in the context of POMDPs it does not take into account the possible improvement of a backup over other belief points as well. It is quite likely that executing a backup that improves a region of the belief space rather than a single belief point may have better influence over the convergence of the value function. Thus, future work should examine other possible heuristic functions that take this issue into account. The Bellman error is also expensive to compute, forcing us to estimate only a sampled subset of the belief points – this was very noticeable in our experiments. This implies that cheaper alternatives that lead to similar quality of backup selection may lead to algorithms that are an order of magnitude faster than current algorithms.

Another possible direction for future research is the choice of belief points [3] different algorithms use. Point-based algorithms use different methods for selecting belief points, and better techniques can probably enhance the performance of these algorithms.

References

1. R. I. Brafman. A heuristic variable grid solution method for pomdps. In *AAAI'97*, 1997.
2. A. R. Cassandra, M. L. Littman, and N.L. Zhang. Incremental pruning: A simple, fast, exact method for partially observable markov decision processes. In *UAI'97*, pages 54–61, 1997.
3. M. Izadi, D. Precup, and D. Azar. Belief selection in point-based planning algorithms for pomdps. In *AI'06*, 2006.
4. M. L. Littman, A. R. Cassandra, and L. P. Kaelbling. Learning policies for partially observable environments: Scaling up. In *ICML'95*, 1995.
5. W. S. Lovejoy. Computationally feasible bounds for partially observable markov decison processes. *Operations Research*, 39:175–192, 1991.
6. S. Paquet, L. Tobin, and B. Chaib-draa. Real-time decision making for large pomdps. In *AI'2005*, 2005.
7. J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *IJCAI*, August 2003.
8. P. Poupart and C. Boutilier. VDCBPI: an approximate scalable algorithm for large POMDPs. In *NIPS 17*. MIT Press, 2004.
9. R. Smallwood and E. Sondik. The optimal control of partially observable processes over a finite horizon. *Operations Research*, 21, 1973.
10. T. Smith and R. Simmons. Heuristic search value iteration for pomdps. In *UAI 2004*, 2004.
11. T. Smith and R. Simmons. Point-based pomdp algorithms: Improved analysis and implementation. In *UAI 2005*, 2005.
12. M. T. J. Spaan and N. Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *JAIR*, 24:195–220, 2005.
13. D. Wingate and K. D. Seppi. Prioritization methods for accelerating mdp solvers. *JMLR*, 6:851–881, 2005.