Noname manuscript No.
(will be inserted by the editor)

A Survey of Point-Based POMDP Solvers

Guy Shani · Joelle Pineau · Robert Kaplow

Received: date / Accepted: date

Abstract The past decade has seen a significant breakthrough in research on solving partially observable Markov decision processes (POMDPs). Where past solvers could not scale beyond perhaps a dozen states, modern solvers can handle complex domains with many thousands of states. This breakthrough was mainly due to the idea of restricting value function computations to a finite subset of the belief space, permitting only local value updates for this subset. This approach, known as point-based value iteration, avoids the exponential growth of the value function, and is thus applicable for domains with longer horizons, even with relatively large state spaces. Many extensions were suggested to this basic idea, focusing on various aspects of the algorithm — mainly the selection of the belief space subset, and the order of value function updates. In this survey, we walk the reader through the fundamentals of point-based value iteration, explaining the main concepts and ideas. Then, we survey the major extensions to the basic algorithm, discussing their merits. Finally, we include an extensive empirical analysis using well known benchmarks, in order to shed light on the strengths and limitations of the various approaches.

Keywords Partially observable Markov decision processes · decision-theoretic planning · reinforcement learning

1 Introduction

Many autonomous agents operate in an environment where actions have stochastic effects. In many such cases, the agent perceives the environment through

Guy Shani
Information Systems Engineering, Ben Gurion University, ISRAEL
E-mail: shanigu@bgu.ac.il

Joelle Pineau, Robert Kaplow
School of Computer Science, McGill University, Montreal, QC, CANADA
E-mail: jpineau@cs.mcgill.ca

noisy and partial observations. Perhaps the most common example of this setting is a robot that receives input through an array of sensors (Huynh and Roy, 2009; Hsiao et al, 2007; Atrash et al, 2009). These sensors can provide only partial information about the environment. For example, robotic sensors such as cameras and lasers cannot see beyond walls, and the robot thus cannot directly observe the contents of the next room. Thus, many features of the problem, such as the existence of hazards or required resources beyond the range of the sensors, are hidden from the robot. Other examples of applications where partial observability is prevalent are dialog systems (Williams and Young, 2007), preference elicitation tasks (Boutilier, 2002; Doshi and Roy, 2008), automated fault recovery (Littman et al, 2004; Shani and Meek, 2009), medical diagnosis (Hauskrecht and Fraser, 2000), assisting people with disabilities (Hoey et al, 2010), recommender systems (Shani et al, 2005), and many more.

For such applications, the decision-theoretic model of choice is a partially observable Markov decision process (POMDP). POMDPs provide a principled mathematical framework to reason about the effects of actions and observations on the agent's perception of the environment, and to compute behaviors that optimize some aspect of the agent's interaction with the environment.

Up until recently, researchers attempting to solve problems that naturally fitted the POMDP framework tended to choose other, less expressive models. The main reason for compromising on the accurate environment modeling was the lack of scalable tools for computing good behaviors. Indeed, slightly more than a decade ago, POMDP researchers still struggled to solve small toy problems with a handful of states, or used crude approximations that typically provided low quality behaviors.

Over the past decade, a significant breakthrough has been made in POMDP solving algorithms. Modern solvers are capable of handling complex domains with many thousands of states. This breakthrough was due in part to an approach called point-based value iteration (Pineau et al, 2003a), which computes a value function over a finite subset of the belief space. A point based algorithm explores the belief space, focusing on the reachable belief states, while maintaining a value function by applying the point-based backup operator.

The introduction of this approach paved the way to much additional research on point-based solvers, leading to the successful solution of increasingly larger domains (Smith and Simmons, 2005; Shani et al, 2008a; Spaan and Vlassis, 2005; Kurniawati et al, 2008; Poupart et al, 2011). Point-based solvers differ in their approaches to various aspects of the point-based value iteration procedure, mainly their method for selecting a core subset of the belief space over which to apply dynamic programming operations, as well as on the order by which the value at those beliefs are updated.

This goal of this paper is to provide a comprehensive overview of point-based value iteration approaches. It can be used to provide a thorough introduction to the topic, as well as a reference for the more advanced reader. The paper is meant as a practical guide towards the deployment of point-based POMDP solvers in practice. Thus we report on extensive experiments covering contrasting benchmark domains, in order to shed light on the empirical properties of various algorithmic aspects. While many of the point-based algorithms have been subject to empirical comparisons in the past, one of the strengths of the analysis presented in this paper is that it systematically compares specific algorithmic components (whereas full algorithms typically differ in many ways). We anticipate that our results and discussion will stimulate new ideas for researchers interested in building new, faster, and more scalable algorithms. We do not, however, try in the scope of this survey to propose and analyze new innovative algorithms, or new combinations of existing algorithms that may prove better than existing approaches.

Some of the main conclusions of our analysis include recommendations for which belief collection methods to use on which types of problems, and which belief updating methods are best paired with which belief collection methods. We also present evidence showing that some parameters (including the number of belief points collected between value function updates, and the number of value function updates between belief collection rounds) have little impact on performance of the algorithm. We also present results showing the impact of the choice of initial solution, and the impact of the order of the value updates, all aspects which can make the difference between successfully solving a hard problem, and not finding a solution.

The paper is structured as follows; we begin with a thorough background on POMDPs, the belief-space MDP, and value iteration in belief space. We then introduce the general point-based value iteration algorithm, and discuss its various components. We then move to describing the various approaches to the implementation of these different components and their properties. We describe a set of experiments that shed light on these properties. Finally, we discuss some related issues, point to several key open questions, and end with concluding remarks.

2 Background

In this section we provide relevant background for understanding the point-based value iteration procedure. We begin with describing the Markov decision process (MDP) framework, and then move to POMDPs. We describe the belief space view of POMDPs, and the extension of value iteration from MDPs to POMDPs.

2.1 MDPs

Consider an agent operating in a stochastic environment; the agent influences the world through the actions it executes, attempting to change the world state to achieve a given objective. Let us assume that the environment is Markovian, that is, that the effects of an action (stochastically) depend only on the current world state, and on the action that the agent chooses to execute.

Example 1 (Robotic navigation in a known environment) A robot is required to move from one location to another in a known environment. The robot can execute high level commands, such as turning left or right, or moving forward. These commands are then translated into long sequences of signals to the robot's motors. As the execution of each of these low level signals can be imprecise, the result of applying the entire sequence may be very different from the intention of the robot. That is, the robot may try to turn left by $90°$ and end up turning $120°$, $45°$ or not at all. Also, when moving forward, the robot may actually deviate from its intended direction. The robot must execute actions that will bring it from its current location to the goal location, minimizing the required time for reaching that goal.

2.1.1 Formal MDP definition

Such environments can be modeled as a Markov decision process (MDP) (Bellman, 1957a; Howard, 1960; Sutton and Barto, 1998; Szepesvari, 2009), defined as a tuple $< S, A, T, R >$. where:

- S is the set of environment states. To be Markovian, each state must encompass all the relevant features for making correct decisions. In the example above, the robot's state encompasses the world map, as well as the robot's own location and orientation within that map. In case some pathways may be blocked (e.g. by doors), the state also has to describe whether each pathway is blocked.
- A is a set of actions that the agent can execute. In our case, the robot may move forward, turn right, or turn left. In some cases the actions may be parametric. For example, the robot may $TurnLeft(\theta)$. In this paper, for simplicity, we consider only non-parametric actions.
- T is the stochastic transition function, that is, $T(s, a, s') = \Pr(s_{t+1} = s'|s_t = s, a_t = a)$ — the probability of executing action a from state s at time t and reaching state s' at time $t+1$. We are interested in a time-invariant system, that is, a system where the effects of actions are time-independent. Hence, specifying the time t in the equation above is redundant, and we prefer the simpler notation (s') to denote the next time step $(t+1)$. In the navigation example, the transition function models the possible effects of the actions. For example, given that the robot attempts to turn left $90°$, the transition function captures the probability of ending up in any possible successor state, including both the intended position and unintended ones.
- R is the reward function, modeling both the utility of the current state, as well as the cost of action execution. $R(s, a)$ is the reward for executing action a in state s. A negative reward models a cost to executing a in s. In some cases, the reward may also depend on the state after the actions execution, resulting in a reward function of the form $R(s, a, s')$; this can be translated to the simpler $R(s, a)$ by taking the expectation over the next states. In our navigation example, it may be that the robot gets a

reward for executing a *null* action in the goal location, and that the cost of each movement is the time span or the required energy for completing the movement. A popular alternative to reward function, especially in navigation problems, is a cost function, that has a non-zero cost for every action, except in the goal location. However, the reward functions are more widely used than cost functions in current POMDP literature and we therefore choose to use it in this paper.

In this paper we assume a finite and discrete state space, as well as a finite and discrete action space (and later we make the same assumption for the observation space). While this is the most widely used formalism of MDPs and POMDPs, infinite, or continuous state and action spaces can also be used (see, e.g., (Porta et al, 2006; Brunskill et al, 2008) for continuous POMDPs).

It is sometimes beneficial to add to the definition above a set of start states S_0 and a set of goal state S_G. The set of start states limits the set of states of the system prior to executing any action by the agent, allowing many MDP algorithms to limit the search in state space only to the reachable parts. Goal states are typically absorbing, that is, the agent cannot leave a goal state. It is often convenient to define an MDP where the task is to reach some goal state.

2.1.2 Policies and value functions

The objective of MDP planning is to discover an action-selection strategy, called a policy, defining how the agent should behave in order to maximize the rewards it obtains. There are many types of policies, stochastic or deterministic, stationary or non-stationary (that is, dependent on the current time step t), and so forth. In the time-invariant, infinite-horizon MDP case, there exists a stationary and deterministic optimal policy, and hence we discuss only this type of policy here.

Different optimization criteria can be considered, such as the expected sum of rewards, the expected average reward, and the expected sum of discounted rewards. In the discounted case, we assume that earlier rewards are preferred, and use a predefined *discount factor*, $\gamma \in [0, 1)$, to reduce the utility of later rewards. The present value of a future reward r that will be obtained at time t, is hence $\gamma^t r$. In this paper we restrict ourselves to this infinite horizon discounted reward model, but other models that do not require the discount factor exist, such as indefinite-horizon POMDPs (Hansen, 2007) and goal-directed cost-based POMDPs (Bonet and Geffner, 2009).

In this paper, a solution to an MDP is defined in terms of a *policy* that maps each state to a desirable action — $\pi : S \to A$. In the infinite horizon discounted reward problem, the agent seeks a policy that optimizes the expected infinite stream of discounted rewards:

$$E[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t))|\pi, s_0]. \tag{1}$$

This expectation is known as the *value* of the policy. We seek a policy π^* such that:

$$\pi^* = argmax_{\pi} E[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t))|\pi, s_0]. \tag{2}$$

We can also define the value of a policy that starts at state s as

$$V(s) = E[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t))|\pi, s_0 = s]. \tag{3}$$

A function $V : S \to \mathbb{R}$ that captures some value of every state s is called a *value function.*

2.1.3 Value Iteration

Given any value function V, one may define a greedy policy

$$\pi_V(s) = argmax_{a \in A} R(s, a) + \gamma \sum_{s' \in S} T(s, a, s')V(s'). \tag{4}$$

From this, one can compute a new value function

$$V'(s) = R(s, \pi_V(s)) + \gamma \sum_{s' \in S} T(s, \pi_V(s), s')V(s'). \tag{5}$$

More generally, we can define an operator J over value functions:

$$JV(s) = \max_{a \in A} R(s, a) + \gamma \sum_{s' \in S} T(s, a, s')V(s'), \tag{6}$$

known as the Bellman operator, or the Bellman update (Bellman, 1957b). We can show that J is a contraction operator, and thus applying J repeatedly, starting at any initial value function, converges towards a single unique fixed point

$$V^*(s) = \max_{a \in A} R(s, a) + \gamma \sum_{s' \in S} T(s, a, s')V^*(s'). \tag{7}$$

The greedy policy, $\pi_{V^*} = \pi^*$ is the optimal policy for the given problem (Puterman, 1994).

The process of applying J repeatedly is known as *value iteration.* The process can be terminated when the difference between two successive value functions V and JV is less than $\frac{\epsilon(1-\gamma)}{\gamma}$, thus ensuring that the distance between V^* and JV is less than ϵ, that is, $|V^*(s) - JV(s)| \le \epsilon$ for every state s.

In some cases it is more convenient to define an intermediate state-action value function called the Q-function. We can then write:

$$Q(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s')V(s') \tag{8}$$

$$V(s) = \max_{a \in A} Q(s, a). \tag{9}$$

While value-iteration provably converges to the optimal value function, for the purpose of this paper we are more interested in ϵ-convergence, where a value function V is within ϵ of the optimal value function V^*: $\max_s |V(s) - V^*(s)| \leq \epsilon$. This is sufficient in most cases because such a value function predicts the expected sum of discounted rewards to within ϵ of its true value. For the purpose of this paper we will use the term *convergence* to imply an ϵ-convergence.

2.2 Partial Observability

Many real-world domains do not fit the assumptions of the MDP framework, in particular because the agent cannot directly observe the state of the environment at every time step.

Example 2 (Robotic navigation in a partially observable environment) Returning to Example 1, let us further assume that the navigating robot has laser range sensors that can sense nearby walls. The robot does not know its exact location in the environment, but can reason about it given the sensors' information. Such sensors typically have a few limitations; first, they are unable to give full knowledge of the domain, such as objects in other rooms. Furthermore, these sensors provide noisy input, such as occasionally detecting a wall where no wall exists, or failing to identify a nearby wall.

An agent acting under partial observability can model its environment as a partially observable Markov decision process (POMDP) (Sondik, 1971; Kaelbling et al, 1998). A POMDP is formally defined as a tuple $< S, A, T, R, \Omega, O, b_0 >$, where:

- S, A, T, R are an MDP as defined above, often called the *underlying MDP* of the POMDP.
- Ω is a set of possible observations. For example, in the robot navigation problem, Ω may consist of all possible immediate wall configurations.
- O is an observation function, where $O(a, s', o) = \Pr(o_{t+1}|a_t, s_{t+1})$ is the probability of observing o given that the agent has executed action a, reaching state s'. O can model robotic sensor noise, or the stochastic appearance of symptoms given a disease.

Even though the agent is Markovian with respect to the underlying state, the limited sensor information does not allow the agent to be Markovian with respect to the observations. One option for building a Markovian agent is to use the agent history as its internal state. This history is composed of all the agent's interactions with the environment, starting at time 0, and is typically denoted as $h_t =< a_0, o_1, a_1, o_2, ..., a_{t-1}, o_t >$. Working directly with histories can be cumbersome, but one can alternately work in the space of probability distributions over states, known as *beliefs*.

A belief $b = \Pr(s|h)$ is the probability of being at every state s after observing history h. In the discrete case that we consider here we can think of every $b \in B$ as a vector of state probabilities. That is, for every $s \in S$, $b(s) \in [0,1]$, and $\sum_{s \in S} b(s) = 1$. For cases where the agent is unaware of its initial state, $b_0 = \Pr(s_0)$ provides a distribution over initial states. A POMDP can be defined without an initial belief, but the assumption of some distribution over initial states helps us in establishing the boundaries of the reachable belief space.

The belief changes every time new observations are received or actions are taken. Assuming an agent has current belief state b, then, following the execution of an action a and the observation of signal o, its belief can be updated to:

$$\begin{aligned}
b^{a,o}(s') &= \Pr(s'|b,a,o) && (10)\\
&= \frac{\Pr(s',b,a,o)}{\Pr(b,a,o)} && (11)\\
&= \frac{\Pr(o|s',b,a)\Pr(s'|b,a)\Pr(b,a)}{\Pr(o|b,a)\Pr(b,a)} && (12)\\
&= \frac{O(a,s',o)\sum_{s \in S}\Pr(s'|b,a,s)\Pr(s|b,a)}{\Pr(o|b,a)} && (13)\\
&= \frac{O(a,s',o)\sum_{s \in S} T(s,a,s')b(s)}{\Pr(o|b,a)}, && (14)
\end{aligned}$$

where:

$$\Pr(o|b,a) = \sum_{s \in S} b(s) \sum_{s' \in S} T(s,a,s')O(a,s',o). \qquad (15)$$

The process of computing the new belief is known as a *belief update*.

In fact, in partially observable domains, beliefs provide a sufficient statistic for the history (Sondik, 1971), and thus for deciding on the optimal action to execute. Formally, one can define the belief-space MDP $< B, A, \tau, R_B >$:

– B is the set of all possible beliefs over S. The belief space forms an infinite state space.
– A is the set of possible agent actions as in the original POMDP and the underlying MDP.
– τ is the belief transition function. That is, $\tau(b,a,b')$ is the probability of starting at belief b, executing action a and reaching the new belief b'. We can compute τ as follows: $\tau(b,a,b') = \sum_{o \in \Omega} \Pr(o|b,a)\mathbb{I}(b' = b^{a,o})$, that is, we sum the probabilities of all observations that result in b' being the next belief. While B is infinite, the number of valid successors for a belief is at most $|A| \times |\Omega|$. On the other hand, the number of predecessors of a belief can be unbounded. That is, for some belief b' and action a there can be infinitely many beliefs b that satisfy $\tau(b,a,b') > 0$.
– $R_B(b,a) = \sum_{s \in S} b(s)R(s,a)$ is the expected immediate reward from executing action a at state s. For simplicity of notation we will use $R(b,a)$ instead of $R_B(b,a)$ from now on.

2.2.1 Value Functions for POMDPs

As with the MDP model, we can define the Bellman update operator for the belief-space MDP:

$$JV(b) = \max_{a \in A} R(b,a) + \gamma \sum_{b' \in B} \tau(b,a,b')V(b') = \max_{a \in A} R(b,a) + \gamma \sum_{o \in \Omega} \Pr(o|b,a)V(b^{a,o}). \tag{16}$$

While B is infinite, it has been recognized early on (Sondik, 1978) that the value function for a POMDP, in both the finite and infinite horizon case, can be modeled arbitrarily closely as the upper envelope of a finite set of linear functions, known as α-vectors. Hence we write $V = \{\alpha_1, ..., \alpha_n\}$, the value function defined over the full belief. Using this representation, we can compute the value at a given belief:

$$V(b) = \max_{\alpha \in V} b \cdot \alpha, \tag{17}$$

where $b \cdot \alpha = \sum_{s \in S} b(s) \cdot \alpha(s)$ is the standard inner product operation in vector space.

2.2.2 Value Iteration in Vector Space

The value iteration algorithm over the belief-space MDP can be rewritten in terms of vector operations, and operations on sets of vectors (Sondik, 1978; Cassandra et al, 1997):

$$V' = \bigcup_{a \in A} V^a \tag{18}$$

$$V^a = \bigoplus_{o \in \Omega} V^{a,o} \tag{19}$$

$$V^{a,o} = \{\frac{1}{|\Omega|} r_a + \alpha^{a,o} : \alpha \in V\} \tag{20}$$

$$\alpha^{a,o}(s) = \sum_{s' \in S} O(a,s',o)T(s,a,s')\alpha(s'), \tag{21}$$

where $r_a(s) = R(s,a)$ is a vector representation of the reward function, V is the vector set prior to the backup, V' is the new vector set after the backup, and $V_1 \oplus V_2 = \{\alpha_1 + \alpha_2 | \alpha_1 \in V_1, \alpha_2 \in V_2\}$.

This process is known as *exact value iteration.* In each iteration, the value function is updated across the entire belief space. There are $|V| \times |A| \times |\Omega|$ vectors generated at Equation 21, and computing each of these vectors takes $|S|^2$ operations. In Equation 19 we create $|V|^{|\Omega|}$ new vectors for each action, with a complexity of $|S|$ for each new vector. Hence, the overall complexity of a single iteration is $O(|V| \times |A| \times |\Omega| \times |S|^2 + |A| \times |S| \times |V|^{|\Omega|})$.

In practice, exact value iteration is only feasible for the smallest of problems, since the set of α-vectors grows exponentially with every iteration. As the computational cost of each iteration depends on the number of vectors in V, an exponential growth makes the algorithm prohibitively expensive. To some degree, the sets of α-vectors can be reduced to their minimal form after each stage, resulting in more manageable value functions (Littman, 1996; Cassandra et al, 1997; Zhang and Zhang, 2001). But this is not sufficient for scaling to domains with more than a few dozen states, actions, and observations.

2.2.3 Belief-Value Mapping

The α-vector representation is especially suitable for maintaining a lower bound over the value function, that is incrementally updated. Due to the convergence of value iteration towards the optimal value function, starting with a lower bound, the new vectors will have higher values than previous vectors. Thus, the max operator in Equation 17 will select these new vectors that dominate previous additions to the vector set $\underline{V}$.

On the other hand, when value iteration is initialized using an upper bound, new vectors will typically have a lower value than the currently existing vectors. Adding these vectors to an existing value function, will have no effect, as these vectors will not be selected by the max operator. Thus, incrementally updating an upper bound over the value function represented as a set of α-vector is currently an open problem.

An alternative to the vector set representation of the value function is to use a belief-value mapping, i.e. maintain a value for every belief that is encountered. These mappings are all points on the convex hull of the current value function. Then, one must interpolate the value of beliefs whose mapping is not currently maintained over the convex value function. This can be computed by a linear program:

$$\text{Minimize: } \sum_{i=1}^{|\bar{V}|} w_i \cdot v_i \tag{22}$$

$$\text{Subject to: } b = \sum_{i=1}^{|\bar{V}|} w_i \cdot b_i \tag{23}$$

$$\sum_{i=1}^{|\bar{V}|} w_i = 1 \tag{24}$$

$$w_i \in [0, 1], \tag{25}$$

where $\bar{V} = < b_i, v_i >$ is the set of belief-value mappings that form the upper bound value function. When the size of $\bar{V}$ grows this linear program becomes difficult to solve. Recently, Poupart et al (2011) show that while the value function may change, the belief points that are selected for the interpolation of each unmapped point b usually remains stable. Thus, when one is interested in repeatedly computing the value for a fixed set of unmapped points, it is possible to cache the points selected by the linear program and their weights and compute only the new value as the linear combination of the cached points. In general, when most computations of interpolations are over beliefs that were not previously observed, the linear programming approach is unfeasible.

A number of alternative approximate projections have been suggested in the literature (Hauskrecht, 2000; Armstrong-Crews et al, 2008). Perhaps the most popular approach is the saw-tooth (or jig-saw) approximation to the upper bound, that treats the value function as a set of down-facing pyramids, with their bases at the corners of the belief simplex, and their points at the belief-value mappings. Then, one has only to project the unmapped belief unto the faces of all the pyramids, which can be done using a simple computation.

Algorithm 1 Sawtooth

```
1: $v_b^0 \leftarrow \sum_{b_s \in \bar{V}_{det}} v_s \cdot b(s)$
2: **for each** $< b_i, v_i > \in \bar{V}_{non-det}$ **do**
3:     $v_{b_i}^0 \leftarrow \sum_{b_s \in \bar{V}_{det}} v_s \cdot b_i(s)$
4:     $v_b^i \leftarrow v^0 + (v_i - v_{b_i}^0)(\min_{s:b_i(s)>0} \frac{b(s)}{b_i(s)})$
5: $v_b \leftarrow \min_i v_b^i$
6: **return** $v_b$
```

In Sawtooth (Algorithm 1), the points in $\bar{V}$ are divided into the corner belief points, $\bar{V}_{det} = \{b_s : b_s(s) = 1\}$, i.e. beliefs that are deterministic in a single state, and the rest of the points $\bar{V}_{non-det}$. We first compute the value of the query point b using an interpolation of the corner points only, and then compute the projection of b onto each of the pyramids associated with mapping $< b_i, v_i > \in \bar{V}_{non-det}$ (Hauskrecht, 2000; Smith and Simmons, 2005).

3 Point-Based Value Iteration

An important contribution to POMDP research in the past decade was the introduction of the point-based value iteration algorithm, that allows us to approximately solve large POMDPs rapidly. This approach and most of its numerous variants are introduced in this section. We begin with presenting the basic insight that the point-based approach leverages, and continue with a thorough description of the important variants of this approach.

3.1 Bounding the Size of the Value Function

As discussed above, it is crucial to limit the size of the set of vectors representing the value function when performing value iteration over the belief space MDP. Obviously, there is a trade-off between avoiding the exponential growth in representation size, at the cost of compromising the accuracy of the value function. So we must decide wisely which vectors should be removed.

One simple solution would be to select a set of belief points, and maintain only vectors that are optimal for at least one belief point in the set. This approach was first suggested by Lovejoy (1991), who maintained a regular grid of belief points, and pruned out of V all α-vectors that were not optimal for this belief subset. One downside of such an approach (using regular grids), is that it is highly probable that many of these belief points are not reachable. That is, there is no possible sequence of actions and observations that leads from b_0 to a regular grid point. Hence, we optimize the value function for a set of beliefs that will never be visited during policy execution.

This problem can be overcome by instead collecting a set of reachable belief points, and maintaining the value function only over these points (Hauskrecht, 2000; Pineau et al, 2003a). Collecting a set of such points can be done by applying the belief update procedure starting from the initial belief state. We now have to decide which sequence of actions and observations should be selected in order to collect good sets of belief points, a question that we will address later in detail. But first, we describe how to update the value function at a specific set of points, denoted B.

3.2 Updating the Value Function

When updating the value function at a finite subset of the belief set B, it is not necessary to use a full Bellman backup (Equations 18-21). Instead, we can use a simple manipulation of the value function update procedure to come up with a less expensive solution. Below, we begin with an equation that computes the value at a certain belief point b after a Bellman backup over a given value function V. We show how the computation of this value can be used to efficiently compute the new α-vector that would have been optimal for b, had we ran the complete Bellman backup process. Using the vector notation

$R(b,a) = r_a \cdot b$, we can write:

$$
\begin{aligned}
V'(b) &= \max_{a\in A} r_a \cdot b + \gamma \sum_{o\in\Omega} \Pr(o|b,a) V(b^{a,o}) && (26)\\
&= \max_{a\in A} r_a \cdot b + \gamma \sum_{o\in\Omega} \Pr(o|b,a) \max_{\alpha\in V} b^{a,o}\cdot\alpha && (27)\\
&= \max_{a\in A} r_a \cdot b + \gamma \sum_{o\in\Omega} \Pr(o|b,a) \max_{\alpha\in V} \sum_{s'\in S} \alpha(s') b^{a,o}(s') && (28)\\
&= \max_{a\in A} r_a \cdot b + \gamma \sum_{o\in\Omega:\Pr(o|b,a)>0} \Pr(o|b,a) \max_{\alpha\in V} \sum_{s'\in S} \alpha(s') \frac{O(a,s',o)}{\Pr(o|b,a)} \sum_s b(s) T(s,a,s') \\
&= \max_{a\in A} r_a \cdot b + \gamma \sum_{o\in\Omega:\Pr(o|b,a)>0} \max_{\alpha\in V} \sum_s b(s) \sum_{s'\in S} \alpha(s') O(a,s',o) T(s,a,s') && (29)\\
&= \max_{a\in A} r_a \cdot b + \gamma \sum_{o\in\Omega:\Pr(o|b,a)>0} \max_{\alpha\in V} \sum_s b(s) \alpha^{a,o}(s) && (30)\\
&= \max_{a\in A} r_a \cdot b + \gamma \sum_{o\in\Omega:\Pr(o|b,a)>0} \max_{\alpha\in V} b\cdot\alpha^{a,o}. && (31)
\end{aligned}
$$

where

$$\alpha^{a,o}(s) = \sum_{s'\in S} \alpha(s') O(a,s',o) T(s,a,s'). \tag{32}$$

We can now write a compact backup operation, that generates a new α vector for a specific belief b:

$$backup(V,b) = \operatorname*{argmax}_{\alpha_a^b : a\in A, \alpha\in V} b\cdot\alpha_a^b \tag{33}$$

$$\alpha_a^b = r_a + \gamma \sum_{o\in\Omega} \operatorname*{argmax}_{\alpha^{a,o}:\alpha\in V} b\cdot\alpha^{a,o}. \tag{34}$$

This procedure implicitly prunes dominated vectors twice, at each argmax expression. Thus, we never have to run the costly operations of Equation 18 and Equation 19 that generate an abundance of vectors. Still, one has to generate $|A|\times|O|$ new vectors (all possible $\alpha^{a,o}$). However, $\alpha^{a,o}$ is independent of b and can therefore be cached and reused for backups over other belief points in B.

The complexity of computing Equation 32 is $O(|S|^2)$, and it is done for every $\alpha \in V$, hence computing all $\alpha^{a,o}$ requires $O(|A| \times |\Omega| \times |V| \times |S|^2)$. Computing α_a^b (Equation34) requires the computation of all the relevant $\alpha^{a,o}$, but then the summation and inner products require only $O(|S| \times |\Omega|)$ operations and another $O(|S|)$ operations for adding the reward vector. Finally, the backup operation (Equation 33 requires for each α_a^b another $O(|S|)$ operations for the inner product. Hence, the full complexity of the point-based backup requires $O(|A| \times |\Omega| \times |V| \times |S|^2 + |A| \times |S| \times |\Omega|)$.

However, a full backup for the subset B will not require $|B|$ times the complexity of a single point-based backup, because the $\alpha^{a,o}$ are independent of the current belief point b. Hence, executing a backup for $|B|$ belief points over a single value function V, where we compute every $\alpha^{a,o}$ only once and cache the result, requires only $O(|A| \times |\Omega| \times |V| \times |S|^2 + |B| \times |A| \times |S| \times |\Omega|)$, as compared with the $O(|A| \times |\Omega| \times |V| \times |S|^2 + |A| \times |S| \times |V|^{|\Omega|})$ of a single iteration of the exact backup (Section 2.2.2).

3.3 Executing a Policy following the Point-Based Value Function

With every MDP value function representation, the optimal policy with respect to the value function V can be computed using:

$$\pi_V(s) = \operatorname*{argmax}_a R(s,a) + \gamma \sum_{s' \in S} tr(s,a,s')V(s'). \tag{35}$$

In POMDPs, with a value function defined over the belief space, the optimal policy is:

$$\pi_V(b) = \operatorname*{argmax}_a R(b,a) + \gamma \sum_{o \in \Omega} \Pr(o|b,a)V(b^{a,o}). \tag{36}$$

Computing a policy for the current belief state b using the above equation requires computing all the $|A| \times |\Omega|$ successors of b, with a cost of $|S|^2$ for each successor. Then, computing the value at each successor requires $|S| \times |V|$ operations (using the α-vector representation).

However, if we use the α-vector representation, we can label the vector resulting from the point-based backup operation (Equation 33) with the action that generated it, i.e., the action that resulted in the maximal value. Then, all we need is to find the best α-vector for the current belief state using $\max_{\alpha \in V} b \cdot \alpha$ (Equation 17) and execute the action corresponding to this α-vector is labeled with, with a computation cost of only $|S| \times |V|$ for finding the best action when following V.

3.4 A Generic Point-Based Value Iteration Algorithm

The general point-based approach combines the above two ideas — bounding the value function size by representing the value only at a finite, reachable belief subset, and optimizing the value function using the point-based procedure. Essentially, all point-based algorithms fit into the generic point-based value iteration framework (Algorithm 2).

Algorithm 2 Generic Point-Based Value Iteration

```
1: while Stopping criterion not reached do
2:     Collect belief subset $B$
3:     Update $V$ over $B$
```

The algorithm has two main parts — the collection of the belief subset B, and the update of the value function V. In general, point-based methods differ on the details of how they achieve these two components of the generic algorithm. These techniques are presented in detail in Section 4. In addition, the stopping criterion of the algorithm is typically dependent on the choice of these two components. Many of the algorithms have an anytime nature, continually improving their value function. In these variations, the stopping criterion is time-dependent.

3.5 Initializing the Value Function

An orthogonal, yet important, question that arises in point-based algorithms is the initialization of the value function. As with every value iteration algorithm, it is necessary to begin with some initial function to be updated. It is desirable that the selected initial value function be as close as possible to the optimal V^*, to reduce the number of iterations before convergence.

As we discuss in the next section, some point-based approaches additionally require that the value function be a lower bound on V^*. Finding such a lower bound is relatively easy by setting:

$$R_{min} = \min_{s \in S, a \in A} R(s,a) \tag{37}$$

$$\alpha_{min}(s) = \frac{R_{min}}{1-\gamma} \tag{38}$$

$$V_0 = \{\alpha_{min}\}. \tag{39}$$

This is equivalent to collecting the minimal possible reward in every step, and relies on the convergence of $\sum_{\imath=0..\infty} \gamma^i$ to $\frac{1}{1-\gamma}$. There are, of course, better methods for computing a lower bound that is closer to V^* (Hauskrecht, 1997; Smith and Simmons, 2005), which we also consider in the course of our empirical analysis.

The same considerations apply in the case of the upper bound on the value function, required by some algorithms. As we have explained above, using the vector representation for an incrementally updating upper bound does not seem plausible, and researchers use instead the belief-value mapping representation. Thus, a different initialization strategy is needed, and several such strategies were suggested (Hauskrecht, 1997, 2000). The simplest method is to solve the underlying MDP, and initialize the value function using $\bar{V}(b) = \sum_s b(s)V_{MDP}(s)$, typically known as the Q_{MDP} method (Littman, 1996). The bound can be made tighter, using the fast informed bound (Hauskrecht, 2000), that updates the value function for each state s using the update rule:

$$\bar{Q}_a(s) = R(s,a) + \gamma \sum_o \max_{a'} \sum_{s'} tr(s,a,s')O(a,s',o)\bar{Q}_{a'}(s'). \tag{40}$$

While provably tighter, the fast informed bound computation time for a single value function update is $O(|S|^2 \times |A|^2 \times |\Omega|)$ compared with the Q_{MDP}

initialization that requires only $O(|S|^2 \times |A|)$ computations. In domains with many actions and observations the difference is important, and smart implementation techniques that exploit sparseness in transitions are needed to scale up.

3.6 Parameters Affecting the Generic Algorithm

There are other parameters that control the behavior of the generic point-based algorithm and influence the resulting value function. As most of these parameters are tightly bound to a specific point-based approach, we only discuss them here briefly. These parameters typically induce tradeoffs between the computational effort and the accuracy of the point-based approximation.

A major parameter is the number of belief points in B. In most algorithms, the value update time depends on the size of B, and the accuracy of the value function also typically depends on the number of belief points that were used. As such, there is a tradeoff between a tolerable approximation and the computation time of a value function update. Hence, most point-based approaches control, directly or indirectly, the number of belief points in B.

A second parameter of most point-based approaches is the number of point-based backups in each value function update. In many cases, the value function can be updated without using all the beliefs in B. In other cases, multiple backups over a single belief point in an iteration can be beneficial. Limiting the number of backups reduces the computational burden, while executing more backups may make the value function closer to V^*.

Another important aspect of point-based methods is the removal of dominated vectors. Indeed, many researchers discuss this issue and suggest new methods for detecting dominated vectors. We discuss this aspect further in Section 4.5.

4 Point-Based Algorithms

In this section we review the various point-based methods presented in the literature, from the perspective of the generic point-based algorithm (Algorithm 2). Hence we focus on the approach of each method to these two fundamental questions, namely the selection of B and the update of V given B.

4.1 The original Point-Based Value Iteration (PBVI)

The original approximate point-based method, known as Point-Based Value Iteration (PBVI), was presented by Pineau et al (2003a). PBVI starts with the initial belief state, i.e. $B_0 = \{b_0\}$. Then, for the current belief set B_i, PBVI updates the value function by executing a point-based Bellman backup at every point in the current B_i. That is:

$$V_B^{j+1} = \{backup(b, V_B^j) : b \in B\}, \tag{41}$$

where the backups are executed in no particular order. The process is repeated until $V_B^j = V_B^{j+1}$, or until a predefined number of iterations has been executed.

At this point, to achieve further improvement requires selecting a different belief subset B. The PBVI algorithm does this by selecting for each belief b in B a successor b' that is the most distant from the set B. That is, let L be a distance metric, then we define:

$$|b' - B|_L = \min_{b \in B} |b - b'|_L, \tag{42}$$

and focus on candidate successors generated using forward simulation, thus:

$$b' = \max_{a,o} |b^{a,o} - B|_L. \tag{43}$$

The set of successor points, one b' for each $b \in B_i$, are added into B_i (along with the previous points in B_i) to create the new set B_{i+1}. Experiments by various researchers show little, if any, sensitivity to the distance metric chosen, and L_1 has useful theoretical properties for the convergence of PBVI. The full procedure is formally described in Algorithm 3.

Intuitively, PBVI attempts to select belief points that are spread as evenly as possible across the reachable belief space, trying to span the reachable space, within a given horizon. As PBVI is making locally greedy selections, it may not provide maximal span for a given size of B_i. Nonetheless it can be shown (see (Pineau et al, 2003a) for details) that the error of the value function is bounded by a linear function over the density of the belief subset, i.e. the smaller the gaps between all beliefs, the closer the resulting function is to the optimal value function. Therefore, in the limit and under certain conditions, PBVI converges to the optimal value function.

4.1.1 Alternatives for Expanding B

The belief expansion phase in the original PBVI requires significant computational effort; computing for each belief state all of its successors, requiring $O(|B| \times |A| \times |\Omega|)$ belief update operations at the cost of $O(|S|^2)$ operations each. Followed by computing a distance from each successor belief point to the previous subset B, requiring $O(|B|)$ distance computations for each successor at the cost of $O(|S|)$ operations each. An obvious method for reducing this effort is to sample the set of successors; instead of computing all possible successors of each belief point, we can sample a uniformly and compute only the possible b_a^o for the specific b and a that were chosen, or sample for each action a a single observation o (Pineau et al, 2003a).

The above methods take into account only the coverage over the belief space. We could also leverage the existing value function, to direct the expansion towards points where the policy computed from the current value function would lead the agent. This can be done by greedily selecting an action that is best for the current belief according to this policy, instead of sampling an action randomly (Pineau and Gordon, 2005; Pineau et al, 2006).

The computational cost of the belief set expansion is also largely affected by the multitude of distance metric computations. These computations can be somewhat reduced by using metric trees (Pineau et al, 2003b) — data structures that maintain distance information. Through the use of these structures we can establish, for example, the distance of a belief from the entire set B without explicitly computing its distance from each point in B. However there is overhead to using such structures, and overall performance benefits are not typically large.

The original belief expansion rule of PBVI doubled the size of B at each iteration, by adding for each $b \in B$ a single successor. This exponential growth of B can make the algorithm incapable of dealing with longer problem horizons, as the size of B may be too large to handle when the goal enters the horizon. An alternative strategy, which avoids exponential growth, is to limit the number of added points to a fixed number N, where N is a parameter to the algorithm. When $|B| > N$ we can sample N points from B and compute successors only for these points. Note that a point may be selected more than once in the same iteration or in different iterations. That is, we need to check all the successors of a point $b \in B$.

When we sample the N points, we can sample either inner beliefs (i.e. beliefs that already have a successor in B) or beliefs from the leaves of B. We can bias belief selection towards longer horizons by emphasizing the sampling of these leaves rather than the inner beliefs, following a parameter l (Kaplow, 2010). With probability l, we sample a "leaf", that is, a belief with no successors currently in B. With probability $(1 - l)$ we will sample an "inner belief" — a belief that already has one successor in B. For further refinement, the parameter l could start with a high value, and decay over time, thus offering more flexible balance between finding a long-horizon goal and optimizing the path to that goal.

4.1.2 PEMA

The point-based backup of a value function V given a belief set B, i.e. $V' = \{backup(b, V) : b \in B\}$ is an approximation of the complete backup of V (Equations 18 through 21). One interesting variation of PBVI attempts to find the belief points where the point-based backup is farthest from a complete backup. That is, a point-based backup over all points in B induces a set of α-vectors V_1 while a complete backup will induce a different vector set V_2. PBVI aims at identifying points reachable in one step from B where $|V_1(b) - V_2(b)|$ is maximized. If we add these points to B, the next point-based backup would fix these errors, and thus reduce the distance of the point-based backup from the complete backup.

It has been shown (Pineau and Gordon, 2005) that the error induced by performing a point-based backup at a belief point b' rather than a complete

Algorithm 3 PBVI

```
Function PBVI
1: $B \leftarrow \{b_0\}$
2: while $V$ has not converged to $V^*$ do
3:     $Improve(V, B)$
4:     $B \leftarrow Expand(B)$

Function Improve($V$,$B$)
1: repeat
2:     for each $b \in B$ do
3:         $\alpha \leftarrow backup(b, V)$ //execute a backup operation on all points in $B$ in arbitrary order
4:         $V \leftarrow V \cup \{\alpha\}$
5: until $V$ has converged //repeat the above until $V$ stops improving for all points in $B$

Function Expand($B$)
1: $B_{new} \leftarrow B$
2: for each $b \in B$ do
3:     $Successors(b) \leftarrow \{b^{a,o} | \Pr(o|b,a) > 0\}$
4:     $B_{new} \leftarrow B_{new} \cup \text{argmax}_{b' \in Successors(b)} ||B, b'||_L$ //add the furthest successor of $b$
5: return $B_{new}$
```

backup, denoted $\epsilon(b')$, is at most:

$$\epsilon(b') \leq \sum_{s \in S} \begin{cases} (\frac{R_{max}}{1-\gamma} - \alpha(s))(b'(s) - b(s)) & \text{if } b'(s) \geq b(s) \\ (\frac{R_{min}}{1-\gamma} - \alpha(s))(b'(s) - b(s)) & \text{if } b'(s) < b(s), \end{cases} \tag{44}$$

where b is the point in B closest to b', and α is the vector resulting from the point-based backup of V that is best at b.

We can now compute the estimate:

$$\bar{\epsilon}(b) = \max_{a \in A} \sum_{o \in \Omega} \Pr(o|b,a)\epsilon(b^{a,o}), \tag{45}$$

to choose the point $b \in B$ that has the maximal $\bar{\epsilon}(b)$. Then, add to B its successor $b^{a,o}$ that contributes the most to this error. This heuristic is the PEMA (Point-based Error Minimization Algorithm) variation of PBVI.

This heuristic for expanding the belief subset is more informative than the original distance-based PBVI heuristic as it attempts to approximate as closely as possible the exact value iteration process, whereas PBVI attempted to cover the reachable belief space as closely as possible, ignoring the computed value function.

However, computing $\bar{\epsilon}(b)$ for each $b \in B$ takes $O(|B| \times |A| \times |\Omega|)$ belief updates to compute all the successors (as in PBVI), and for each successor we would compute its closest belief using $O(|B|)$ distance computations. Finally, we would compute each error in $O(|S|)$ operations. Thus in general this approach is much slower than the original PBVI, though it can yield better results (Pineau et al, 2006) in some domains.

4.2 Perseus

The original PBVI algorithm and its variants focus on the smart expansion of B, both because B governs the size of V in PBVI, and because expanding B requires significant effort. However, if we could collect a large B effortlessly, and still compute a compact value function, we could leverage the efficiency of point-based Bellman backups, without performing expensive belief selection steps.

Spaan and Vlassis (2004, 2005) highlight the intuition that one can be non-selective in constructing B if V is updated smartly in their Perseus algorithm. The Perseus algorithm starts with running trials of random exploration in belief space. At each step of a trial, an action and observation are sampled, and a new belief is computed and added to B. The trials continue until a (relatively) large number of points are collected. Similar use of random forward simulation to collect belief points was explored, though with less impact, in the early days of approximate POMDP solvers (Hauskrecht, 2000; Poon, 2001; Lovejoy, 1991).

Once B has been collected, Perseus proceeds to the value function computation phase. The computation is done iteratively. Each iteration begins with a set $B' = B$ and a new empty value function $V' = \phi$. One belief point $b \in B'$ is selected uniformly at random, and used to compute a new vector $\alpha_{new} = backup(b, V)$. If $\alpha_{new} \cdot b > V(b)$, then α_{new} is added into V', and we remove from B' all b' such that $\alpha_{new} \cdot b' > V(b')$. That is, all belief points whose value was improved by α_{new} will not be backed up at the current iteration. If $\alpha_{new} \cdot b \leq V(b)$, we add $\alpha_{old} = \text{argmax}_{\alpha \in V}\, \alpha \cdot b$ into V'. The iteration ends when $B' = \phi$, and V is set to V'. The full procedure is outlined in Algorithm 4.

This value function update procedure guarantees that even though many beliefs are not backed up at each iteration, the value for every point $b \in B$ gets closer to $V^*(b)$ with every iteration. Still, since many belief points will not get backed up, the value function may remain relatively small. Note however that in this algorithm, it is assumed that if $\alpha_{new} \cdot b > V(b)$, then $|\alpha_{new} \cdot b - V^*(b)| < |V(b) - V^*(b)|$. This is only true if we initialize V to a lower bound on V^*. Therefore, unlike PBVI, Perseus requires a lower bound initialization.

Perseus has two key advantages; the belief collection step is fast, and the value function update tends to create value functions with a number of α-vectors much smaller than $|B|$. Thus, Perseus can collect a set of belief points much larger than PBVI, and has thus the potential to cover more of the reachable belief space.

There are a number of potential disadvantages to Perseus, though; first, the random belief gathering assumes that it is relatively simple to obtain good belief points. That is, that a random exploration will eventually encounter most of the same points as the optimal policy. In many goal-directed domains (Bonet and Geffner, 2009), where actions must be executed in a specific order, the random exploration assumption may be problematic. Second, the uninformed selection of beliefs for backups may cause the value function to be updated very slowly, and with many unneeded backups that do not contribute to the convergence. For example, in goal-directed domains, values typically get gradually updated from beliefs closer to the goal to beliefs farther from it, until finally the value of b_0 is updated. The best backup order in these cases is by moving in reverse order along a path in belief space that the optimal policy may follow. Randomly picking beliefs, even along such an optimal path, could substantially slow down convergence (Shani et al, 2008a).

Finally, a practical limitation in large domains with long trajectories is that Perseus will need to collect a very large set of beliefs, causing the set B to become too large to maintain in memory. An iterative version of Perseus that collects a reasonably sized belief set, improves it, and then collects a second set, may be able to overcome this problem. In this case, we could also leverage the policy that was already computed, instead of a random policy, for future collection of belief points.

Algorithm 4 Perseus

```
Function Perseus
 1: $B \leftarrow RandomExplore(n)$
 2: $V \leftarrow PerseusUpdate(B, \phi)$

Function RandomExplore(n)
 1: $B \leftarrow \phi$
 2: $b \leftarrow b_0$
 3: repeat
 4:
      choose a random successor of $b$
 5:   Choose $a \in A$ randomly
 6:   Choose $o \in \Omega$ following the $\Pr(o|b, a)$ distribution
 7:   Add $b^{a,o}$ to $B$
 8:   $b \leftarrow b^{a,o}$
 9: until $|B| = n$

Function PerseusUpdate(B,V)
 1: repeat
 2:   $B' \leftarrow B$
 3:   $V' \leftarrow \phi$
 4:   while $B' \neq \phi$ do
 5:     // Choose an arbitrary point in $B$ to improve
 6:     Choose $b \in B'$
 7:     $\alpha \leftarrow backup(b, V)$
 8:     if $\alpha \cdot b \geq V(b)$ then
 9:       // Remove from $B$ all points whose value was already improved by the new $\alpha$
10:       $B' \leftarrow \{b \in B' : \alpha \cdot b < V(b)\}$
11:       $\alpha_b \leftarrow \alpha$
12:     else
13:       $B' \leftarrow B' - \{b\}$
14:       $\alpha_b \leftarrow \text{argmax}_{\alpha \in V}\, \alpha \cdot b$
15:     $V' \leftarrow V' \cup \{\alpha_b\}$
16:   $V \leftarrow V'$
17: until $V$ has converged
```

Informed Selection of Backup Order

The Perseus approach ensures that there will be an improvement in the value function at each iteration, while preserving a small value function. However, as we argue above, in some cases the random choice of points at which to backup may result in a slow convergence rate. Some researchers have suggested exploiting additional information for selecting the next belief point at which to execute a Bellman backup.

First, one can leverage information about the value function update procedure itself in order to select the belief at which to apply the backup (Shani et al, 2008a). The Bellman update procedure (Equation 33) suggests that the value of a belief will change only if the value of its successor points is modified. Hence, a reasonable approach would be to update those belief states whose successor value changed the most. In the MDP domain, this approach is known as prioritized value iteration (Wingate and Seppi, 2005), and has been shown to converge much faster than arbitrary ordering of backups. In the POMDP case, implementing the prioritized approach is much harder, because beliefs can have an infinite number of predecessors, making the backward diffusion of values difficult, and because, with the α-vector representation, each backup may introduce a new vector that changes the value of many states.

An alternative approach leverages the structure of the underlying MDP state space, in particular for domains where the state space has a DAG structure, or when the states can be clustered into larger components that have a DAG structure (Bonet and Geffner, 2003; Dai and Goldsmith, 2007). In such cases, we can compute the value function over the components in reversed DAG order. We start with the leaves of the graphs and compute the value of a component only if all its successors have already been computed. The same idea has been transformed into the POMDP world, and can be applied also to domains that are not strictly DAGs (Dibangoye et al, 2009).

Finally, one can combine both information about the value function and information about the structure of the state space. We could use the optimal policy of the MDP to cluster states into layers, such that states that are closer to the goal, following the MDP policy belong to a higher level than states that are farther than the goal. For example, states that can get to the goal using a single action, following the MDP optimal policy may belong to the last layer, while states that require two actions to get to the goal belong to the layer before last. Then, we can associate belief points to layers given their probability mass over states in specific layers. We then iterate over the layers in reversed order, selecting belief points to update only from the active layer (Virin et al, 2007).

All these approaches still suffer to some degree from the same limitations as Perseus, namely, the random collection of belief points, and the difficulty of maintaining a large set of belief points. In fact, as all these methods compute additional information, such as the belief clusters, the iterative approach suggested towards the end of the previous section becomes harder to implement.

4.3 Heuristic Search Value Iteration (HSVI)

The randomization aspects of Perseus are well suited to the small to mid-sized domains that were most prevalent a decade ago. Yet, as we argue above, applying Perseus to more complex domains may fail, because of its unguided collection of belief points and the random order of backups over the collected beliefs. For value function updates, there exists a simple, yet effective, heuristic — maintain the order by which belief points were visited throughout the trial, and back them up in a reversed order. As the value of a belief is updated based on the value of its successors (see Equation 26), updating the successors prior to updating the current belief may accelerate the convergence of value iteration. This insight is a prime component of the Heuristic Search Value Iteration (HSVI) approach (Smith and Simmons, 2004, 2005).

Another important contribution of HSVI is in the heuristic it uses to focus belief point collection on the points that are most relevant to the value function. This is achieved using the bound uncertainty heuristic; by maintaining both an upper and lower bound over the value function, we can consider the difference between the bounds at a specific belief as the uncertainty at that belief. The higher this distance, the more uncertain we are about the optimal value at that belief. Assuming the points considered are successors of the initial belief, b_0, then reducing these bounds contributes directly to reducing the bounds at b_0. When the distance between the bounds has dropped below some threshold on b_0, we can claim that value iteration has converged. This convergence is different than the guaranteed convergence of exact value iteration, where the value function is within a certain distance for every possible belief state. Still, in a discount-based POMDP with a given initial belief, one can argue that we only care about the expected discounted sum of rewards that could be earned starting at the given b_0, and hence that value iteration has converged when this expected discounted sum has been accurately computed.

Formalizing this idea, HSVI greedily selects successors so as to maximize the so-called excess uncertainty of a belief:

$$excess(b,t) = (\overline{V}(b) - \underline{V}(b)) - \frac{\epsilon}{\gamma^t}, \tag{46}$$

where $\overline{V}$ is the upper bound and $\underline{V}$ is the lower bound on the value function, ϵ is a convergence parameter, and t is the depth of b (i.e. number of actions from b_0 to b) during the trial.

When selecting actions during the trial, HSVI chooses greedily based on the upper bound. The reason is that as the value function is updated, the upper bound is reduced. Therefore, an action that currently seems optimal, based on the upper bound, can only have its value reduced. Eventually, if the action is suboptimal, its value will drop below the value of another action. Choosing greedily based on the lower bound will have the opposite effect. The action that currently seems optimal can only have its value increase. Hence, if the current action is suboptimal we will never know that if we do not try other actions.

Algorithm 5 HSVI

```
Function HSVI
1: Initialize $\underline{V}$ and $\bar{V}$
2: while $\bar{V}(b_0) - \underline{V}(b_0) > \epsilon$ do
3:     $BoundUncertaintyExplore(b_0, 0)$

Function BoundUncertaintyExplore($b$, $t$)
1: if $\bar{V}(b) - \underline{V}(b) > \epsilon\gamma^{-t}$ then
2:     // Choose the action according to the upper bound value function
3:     $a^* \leftarrow \text{argmax}_a\, Q_{\bar{V}}(b, a')$
4:     // Choose an observation that maximizes the gap between bounds
5:     $o^* \leftarrow \text{argmax}_o(\Pr(o|b, a^*)(\bar{V}(b^{a,o}) - \underline{V}(b^{a,o}) - \epsilon\gamma^{-(t+1)}))$
6:     $BoundUncertaintyExplore(b^{a^*,o^*}, t + 1)$
7:     // After the recursion, update both bounds
8:     $\underline{V} = \underline{V} \cup backup(b, \underline{V}))$
9:     $\bar{V}(b) \leftarrow J\bar{V}(b)$
```

Once the trial has reached a belief b at depth t such that $\overline{V}(b) - \underline{V}(b) \leq \frac{\epsilon}{\gamma^t}$, the trial terminates, because the potential change to the value at b_0 from backing up the trajectory is less than ϵ. Then, the beliefs along the trajectory explored during the trial are backed up in reverse order. That is, we first execute a backup on the belief that was visited last. Both the upper and lower bounds must be updated for each such belief. An α-vector that results from the backup operation is added to the value function $\underline{V}$, and a new belief-value mapping is added to the upper bound. The HSVI approach is outlined in Algorithm 5.

This incremental method for updating V allows the computation of backups over many belief points, without the requirement to maintain all those beliefs in memory. However, the incremental update poses a new challenge: as opposed to PBVI that has a single vector per observed belief point $b \in B$ at most, HSVI may visit the same belief point in many trials, or even several times during the same trial, and add a new vector for each such visit. It is also likely that vectors that were computed earlier, will later become dominated by other vectors. It is useful to remove such vectors in order to reduce the computation time of the backup operator.

An advantage of the bound uncertainty heuristic is that HSVI provably converges to the optimal value function. As HSVI seeks the beliefs where the gap between the bounds is largest, and reduces these gaps, it reduces at each iteration the gap over the initial belief state b_0. Once the gap over b_0 drops below ϵ, we can be certain that the algorithm has converged. Moreover, HSVI guarantees termination after a finite number of iterations, although this number is exponential in the maximal length of a trial.

The upper bound is represented using belief-value mappings, and projections are computed using the Sawtooth approximation (Algorithm 1).

In later work, Smith and Simmons suggest revising the selection of the best current α-vector — $\text{argmax}_{\alpha \in V}\, b \cdot \alpha$ — to only consider α-vectors that were computed for beliefs that had the same set of non-zero entries (Smith and Simmons, 2005). They introduce *masks* to rapidly compute which vectors are eligible for selection.

Breadth First Search

While HSVI uses a depth-first search, it is also possible to use a breadth-first search, still focused on beliefs where the gap between the bounds contributes the most to the gap over the initial belief state b_0. Poupart et al (2011) suggest in their GapMin algorithm (Algorithm 6) to use a priority queue to extract the belief that contributes the most to the initial belief gap. This belief is then expanded, i.e., its successors are computed and added to the priority queue. In expanding the belief, only the best action for the upper bound is considered, as in HSVI. Unlike HSVI, however, all possible observations are expanded. After collecting the belief points, GapMin updates the lower bound using the PBVI improvement (Algorithm 3).

Algorithm 6 GapMin

```
Function GapMin
 1: Initialize $\underline{V}$ and $\bar{V}$
 2: while $\bar{V}(b_0) - \underline{V}(b_0) > \epsilon$ do
 3:     $B \leftarrow CollectSuboptimalBeliefs(\underline{V}, \bar{V})$
 4:     $Improve(\underline{V}, B)$
 5:     $Update(\bar{V})$

Function CollectSuboptimalBeliefs($b$, $t$)
 1: $B \leftarrow \phi$
 2: $pq \leftarrow$ the empty priority queue
 3: $pq.Insert(< b_0, \bar{V}(b_0) - \underline{V}(b_0), 1, 0 >)$
 4: while $pq \neq \phi$ do
 5:     $< b, score, prob, depth > \leftarrow pq.ExtractMaxScore()$
 6:     // Choose an action following the upper bound
 7:     $a^* \leftarrow \text{argmax}_a \, Q_{\bar{V}}(b, a')$
 8:     // If the upper bound at the new point can be improved, add the mapping to $\bar{V}$
 9:     if $\bar{V}(b) - Q_{\bar{V}}(b, a^*) > tolerance$ then
10:         $\bar{V} \leftarrow \bar{V} \cup \{b, Q_{\bar{V}}(b, a^*)\}$
11:     // If the lower bound at the new point can be improved, add $b$ to $B$
12:     $\alpha \leftarrow backup(\underline{V}, b)$
13:     if $\alpha \cdot b - \underline{V}(b) > tolerance$ then
14:         $B \leftarrow B \cup b$
15:     // Add all the successors of the $b$ with a gap above the threshold to the priority queue
16:     for each $o \in \Omega$ do
17:         $gap \leftarrow \bar{V}(b_{a^*,o}) - \underline{V}(b_{a^*,o})$
18:         if $\gamma^{depth+1} \cdot gap > tolerance$ then
19:             $prob_o \leftarrow prob \cdot \Pr(o|b, a)$
20:             $score \leftarrow prob_o \cdot \gamma^{depth+1} \cdot gap$
21:             $pq.Insert(< b_{a^*,o}, score, prob_o, depth + 1 >)$
```

Another important contribution of the GapMin algorithm is with respect to the upper bound representation and update. Poupart et al (2011) observe that the belief points used to interpolate the upper bound value for a given belief remain stable, even when their upper bound value changes. Thus, by caching these interpolation points, one can compute the interpolation very rapidly. The *Update* procedure for the upper bound in Algorithm 6 defines a POMDP called the Augmented POMDP that uses the interpolation points to define a tighter upper bound with a smaller set of belief-value mappings. In the latter sections of this paper we do not explore various representation and interpolation techniques for the upper bound, although this is certainly of interest and may significantly affect point-based algorithms.

Other Heuristics for Forward Search

HSVI uses the bound uncertainty heuristic in choosing trajectories, but other heuristics could be considered. The Forward Search Value Iteration (FSVI) algorithm (Shani et al, 2007) leverages a different insight; it constructs trajectories in belief space using the best action following the policy of the underlying MDP. Such trajectories typically focus the belief search towards areas of high expected reward, and do so quickly by ignoring partial observability. Once a trajectory in belief space has been acquired, value updates are applied in reverse order of belief collection to construct an improved value function.

An obvious limitation of this heuristic is that it does not select actions that can reveal information about the true state of the environment. For example, if collecting information requires a sequence of actions that moves the agent away from the goal/rewards (Geffner and Bonet, 1998), FSVI will not attempt these trajectories, and will not learn how the information can affect the policy. Still, FSVI will evaluate these actions during the point-based backup operation. Thus, if information can be obtained using some action in the current belief state, this action will be evaluated and may become a part of the policy. The FSVI approach is outlined in Algorithm 7.

Algorithm 7 FSVI

```
Function FSVI
1: Initialize $V$
2: while $V$ has not converged do
3:    Sample $s_0$ from the $b_0$ distribution
4:    // Compute a trajectory assuming that $s_0$ is the initial state
5:    MDPExplore($b_0, s_0$)

Function MDPExplore($b, s$)
1: if $s$ is not a goal state then
2:    // Choose an action following the MDP policy for the current sampled state $s$
3:    $a^* \leftarrow \text{argmax}_a Q^{MDP}(s, a)$
4:    // Sample the next state and belief state
5:    Sample $s'$ from $T(s, a^*, *)$
6:    Sample $o$ from $O(a^*, s', *)$
7:    MDPExplore($b^{a^*,o}, s'$)
8: $V \leftarrow V \cup backup(b, V)$
```

Another interesting heuristic suggests restricting the set B only to points that are visited during the execution of an optimal policy is sufficient to construct an optimal policy (Kurniawati et al, 2008). Therefore, executing backups on belief points that are not visited during the execution of an optimal policy can be redundant. We can thus ignore all belief points that clearly will not be visited under an optimal policy. Kurniawati et al. show how we can establish that certain belief states will not be visited following an optimal policy without actually computing that optimal policy, and prune these beliefs from B.

While computing the optimal policy is the problem that we set out to solve to begin with, we can use the upper and lower bounds of HSVI in order to establish that certain actions will never be taken. The SARSOP algorithm maintains a belief tree, with b_0 as its root, and prunes out subtrees that will never be visited under the optimal policy (Kurniawati et al, 2008); given a lower and upper bound on the value function, we can sometimes discover that an action a is not optimal in a certain belief state b, because $\bar{Q}(b,a) < \underline{Q}(b,a')$ for some other action a'. In these cases, the entire subtree below the execution of a in b can be pruned out.

SARSOP also introduces a trial termination criterion that emphasizes longer trials, called selective deep sampling. Under this criterion, the trial continues when it is likely to lead to improvements in the lower bound value at belief points earlier in the search. Such improvements can be predicted using a mechanism to estimate the optimal value $V^*(b)$ at the current belief point. SARSOP introduces a clustering based algorithm for providing estimates of V^*.

Bonet and Geffner (2009) show that discount-based POMDPs can be easily translated into cost-based goal-oriented POMDPs, where there is an observable goal state that has zero cost and all other states incur a non-zero cost for every action. In such a POMDP one can use the well known RTDP algorithm (Barto et al, 1995) which provably converges to the optimal value function. An opportunity that was not explored thus far is to use RTDP as a forward-search heuristic for gathering belief points.

4.4 Related Methods

Several ideas related to the point-based concept have arisen in literature. We review such ideas in this section, for the purpose of offering a comprehensive review of the topic. However we do not empirically evaluate these methods in the latter sections of the paper. Nevertheless, these ideas may be useful in some domains.

Izadi et al (2005) suggest a novel strategy for choosing B, that leverages intuitions from predictive state representations (PSRs)(Littman et al, 2001). Following the PSR idea of the history-test matrix, they suggest building a history-belief matrix, and compute a set of core beliefs — beliefs that form a basis of the belief simplex. These beliefs form the set B that will be used to compute the value function. The worst-case error of the value function using this method for selecting B is better than the PBVI approach of covering the belief space as closely as possible. A downside of this method is that computing the core beliefs is non-trivial, and the construction of the matrix must rely on heuristics that no longer guarantee better performance. Even given these approximations, the computation of the matrix is still expensive and difficult to scale.

In another paper, Izadi et al (2006) suggest emphasizing the selection of belief points that are reachable in fewer steps from b_0, because due to discounting, these beliefs may have a greater influence on the value at b_0. In addition, they suggest favoring expansion of beliefs that have extreme (either high or low) values since those are likely to have a larger effect on the convergence of the value function.

Policy iteration using a finite state controller representation of a policy is another approach to solving POMDPs (Hansen, 1998; Poupart and Boutilier, 2003). Policy iteration iterates between two phases: a policy evaluation phase and a policy improvement phase. In the policy evaluation phase the value of the current policy must be computed. This can be done using a point-based approach, where the backup operation is restricted to a specific action — the action dictated by the current policy (Ji et al, 2007). This method has shown very promising empirical results.

Armstrong et al. suggest improving the convergence of point-based algorithms by adding a policy improvement phase motivated again by policy iteration (Armstrong-Crews et al, 2008). They show that both the upper bound and lower bounds, when described by hyperplanes, form an MDP. Thus, they suggest a value iteration procedure over these MDPs that improves the bounds towards V^*. The improvement uses the belief points in B, and can therefore be considered a variant on point-based value function improvement.

Scaling up of POMDP solvers can also be achieved through the relatively new area of parallel and distributed computing. Modern systems have multiple processors on each machine, and large distributed systems are quickly becoming popular for performing heavy computations. The adjustment of POMDP algorithms to these new computation models is probably needed for scaling to real-world problems. Preliminary results show that such point-based algorithms can benefit from multi-core machines (Shani, 2010). Creating point-based algorithms for the more scalable GPU architecture and to large distributed environments remains an open problem.

In structured domains, POMDPs are best described through factored models, which can offer a more compact representation of the problem. Point-based algorithms have been applied to factored POMDPs, using algebraic decision diagrams for α-vector representation (Poupart, 2005; Shani et al, 2008b; Sim et al, 2008). In some domains, further scaling can be achieved through more compact representations such as relational or first-order POMDPs (Sanner and Kersting, 2010; Wang and Khardon, 2010).

4.5 Reducing the Size of V

One core motivation of the point-based approach is the attempt to maintain a compact representation of the value function. As the complexity of the point-based backup, which lies at the core of all point-based algorithms, depends on the number of α-vectors in V, keeping this number low is key to scaling up to longer horizons and larger domains. The inability to maintain small value functions is perhaps the main reason for the failure of exact methods to scale up beyond small domains.

PBVI and Perseus maintain a bounded value function where, at each iteration, the value function has at most $|B|$ vectors. Trial-based algorithms such as HSVI and FSVI that incrementally add new vectors to the value function no longer maintain this property. For the latter set of algorithms, reducing the size of V becomes an important issue. Even for PBVI, reducing the size of V is always desirable, because the complexity of the point-based backup operation depends on $|V|$.

A simple method for pruning vectors is to remove pointwise dominated vectors. We say that a vector α_1 is pointwise dominated by a vector α_2 if for each $s \in S$, $\alpha_1(s) \leq \alpha_2(s)$. Finding pointwise dominated vectors is relatively easy. However, a vector may be dominated by a set of vectors, but not by any single vector. These cases can be detected by a linear program that tries to find a witness for every vector α — a belief point b for which α is optimal, that is, $\alpha = \text{argmax}_{\alpha' \in V}\, b \cdot \alpha'$ (see, e.g., Cassandra et al (1997)). If no witness can be found (the linear program has no solution), then we can prune out α. However, solving this linear program is computationally intensive, and there is currently no known fast method for pruning vectors. An important property of point-based algorithms is the ability to prune vectors that have no witnesses in B. When B is relatively small, and we do not incrementally add vectors to V, this process can be efficient. However, for larger belief sets, and incremental algorithms such as HSVI, this process is no longer practical.

Several other heuristics were suggested for pruning vectors; in a trial-based approach, Armstrong-Crews et al (2008) suggest a pruning technique based on the points that were observed during the execution of the current policy. Pruning all vectors that are not used by these beliefs may cause the value function to degrade. However, maintaining all the vectors that are used by either points that were observed (backed up) or their successor beliefs ensures that the value function still monotonically improves in a sense. Still, computing this pruning strategy requires that we maintain all the beliefs that were backed up and their successors, which may become a burden in larger domains.

A more focused heuristic prunes more aggressively by keeping only vectors that are used in the execution of the currently best policy. When executing a policy, we iteratively update a belief and compute the best vector for the current belief. Removing all vectors that are never selected in this process will not modify the currently optimal policy, even though it may change the value function. Running multiple such trials may consume too much computation power in regular architectures, but in multi-core environments, a single core can be dedicated to executing trials and pruning vectors that were not observed in these trials (Shani, 2010).

Kurniawati et al. use the same idea — removing α-vectors that are not optimal for any belief point observed throughout the execution of the optimal policy (Kurniawati et al, 2008). As we explained above, SARSOP identifies belief sets that will never be visited during the execution of an optimal policy, and can ignore these points when deciding which vectors have witnesses in B.

In general, even though point-based methods benefit from rapid pruning techniques, pruning is not an essential part of point-based methods. One can always limit the growth of the value function by limiting the size of B. Typically, there is a choice to be made between either spending more resources deciding which belief points to collect, or collecting a larger set of belief points and spending time afterwards pruning the value function.

5 Empirical Evaluation Methodology

In the previous section we have reviewed the majority of point-based algorithms and their derivatives. Each algorithm employs ideas that make it more suitable for certain features of a domain. Researchers or practitioners interested in solving specific problems need to choose an appropriate algorithm for the properties of their domain. Indeed, it is not the case that a single algorithm proves best in all possible domains, and different choices of algorithms may result in substantially different performance. It is thus important to show how the properties of various domains influence the performance of point-based algorithms.

In our empirical evaluation, we aim to characterize the performance of different algorithmic approaches in different domains. We chose a set of well-known benchmark domains that illustrate different properties that appear in real-world problems. We are interested in understanding what properties of a domain can be tackled by what algorithmic approach. Thus, in this section we will not report an exhaustive execution of all algorithms over all domains, but rather a set of experiments that explore properties that are broadly representative of contrasting real-world domains.

Many algorithms employ a set of loosely coupled innovations. We would like to evaluate how each of these ideas is appropriate for various domain properties. As such, we implement independently the core algorithmic components, in particular the choice of belief point selection method, and the choice of value function update method. We then evaluate each variation of these components independently. Again, this is in contrast to standard empirical evaluations which report results for many (full) algorithms.

The advantage of organizing the empirical analysis in this manner are many; first, we highlight the relation between the different algorithmic methods. Second, we gain better understanding of the empirical advantages and disadvantages of the various components. Third, we provide evidence-based recommendations on which methods are applicable for specific classes of domains.

This section describes how we conducted our empirical evaluation. First, we review the algorithmic methods included in the evaluation. Then we discuss the experimental domains used throughout the analysis, and finally we outline the empirical measure used to compare the various methods.

5.1 Core Algorithmic Components

As discussed above, the two key components of modern point-based POMDP solvers are the belief collection and value update methods. The algorithms presented in Section 4 differ on one or more of these components. Table 1 provides a detailed mapping between the algorithms — as originally published, and as reviewed in Section 4 — and their collection and update methods. We define each algorithmic component in detail below.

Algorithm	COLLECT	UPDATE
PBVI (Pineau et al, 2003a)	L_1 Norm	Full Backup
Perseus (Spaan and Vlassis, 2005)	Random	Perseus Backup
HSVI (Smith and Simmons, 2005)	Bound Uncertainty (depth-first)	Newest Points Backup
GapMin (Poupart et al, 2011)	Bound Uncertainty (breadth-first)	Full Backup
PEMA (Pineau and Gordon, 2005)	Error Minimization	Full Backup
FSVI (Shani et al, 2008a)	MDP Heuristic	Newest Points Backup

Table 1 An outline of a variety of point-based solvers, where we identify the associated collection and updating methods. Some of these methods feature additional components (for example, HSVI's masked α vectors) that were not included in our analysis.

5.1.1 Belief Point Collection Methods

Recall from Algorithm 2 that the set of beliefs, B, is expanded at each iteration to yield a new set of points B_{new}:

$$B_{new} \leftarrow COLLECT(V_t, B, N). \tag{47}$$

The belief collection method may depend on the current value function V_t, and the previous set of beliefs B. The parameter N represents the number of new points to add.

All the belief point collection methods outlined in Table 1 were discussed in Section 4. Still, for the sake of clarity, we include below pseudo-code for the collection steps only, since in some cases minor changes are necessary (compared with the initial full algorithm) to separate the value function update from the belief collection method.

The Perseus algorithm uses Random forward simulation to collect belief points. Algorithm 8 outlines this simple procedure. Whereas Perseus is described in the original publication as collecting (only) one initial batch of beliefs at the start of the algorithm, here we generalize the method to allow collection of new belief points at each collection step, in keeping with the general framework of Algorithm 2.

Algorithm 8 Random forward simulation for collecting belief points (Perseus)

```
Function RandomCollect(N)
1: $B \leftarrow \phi$
2: $b \leftarrow b_0$
3: while $|B| < N$ do
4:    Choose $a \in A$ randomly
5:    Choose $o \in \Omega$ following the $\Pr(o|b,a)$ distribution
6:    Add $b^{a,o}$ to $B$
7:    $b \leftarrow b^{a,o}$
8:    if $b$ is a goal state then
9:       $b \leftarrow b_0$
```

FSVI adopts a more informed approach, where the MDP heuristic guides the collection of belief points. The procedure as implemented for our analysis is presented in Algorithm 9. As with the random exploration belief, we iterate over belief traces until we have accumulated N belief points.

Algorithm 9 Collection of belief points guided by the MDP solution (FSVI)

```
Function FSVICollect(N)
1: Initialize $B \leftarrow \phi$
2: while $|B| < N$ do
3:    Sample $s_0$ from the $b_0$ distribution
4:    MDPExplore($B, b_0, s_0$)
5: return $B$

Function MDPExplore($B, b, s$)
1: if $s$ is not a goal state then
2:    $a^* \leftarrow \text{argmax}_a\, Q^{MDP}(s,a)$
3:    Sample $s'$ from $T(s, a^*, *)$
4:    Sample $o$ from $O(a^*, s', *)$
5:    Add $b^{a^*,o}$ to $B$
6:    if $|B| = N$ then
7:       return
8:    MDPExplore($B, b^{a^*,o}, s'$)
```

HSVI guides the forward exploration based on the bound uncertainty heuristic, that selects successor beliefs where the gap between the bounds is maximized. As the bounds get updated along the trajectory, the forward exploration and value function update are tightly coupled in HSVI. To reduce this dependency we have decided to use the following separation technique — we update only the upper bound, and leave the lower bound point-based updates to the value function update phase (Algorithm 10). This change makes our HSVI-motivated belief collection method quite different from the original HSVI algorithm. Still, we believe that the change is in the spirit of the HSVI algorithm, that uses the upper bound for guiding the search, and the lower bound for the resulting policy.

Algorithm 10 Collection of belief points guided by the bound uncertainty heuristic (HSVI)

```
Function HSVICollect(N)
1: Initialize $B \leftarrow \phi$
2: while $|B| < N$ do
3:   BoundUncertaintyExplore($B, b_0, 0$)
4: return $B$

Function BoundUncertaintyExplore($B$, $b$, $t$)
1: if $\bar{V}(b) - \underline{V}(b) > \epsilon\gamma^{-t}$ then
2:   $a^* \leftarrow \text{argmax}_a\, Q_{\bar{V}}(b, a')$
3:   $o^* \leftarrow \text{argmax}_o(\Pr(o|b, a^*)(\bar{V}(b^{a,o}) - \underline{V}(b^{a,o}) - \epsilon\gamma^{-(t+1)}))$
4:   Add $b^{a^*,o^*}$ to $B$
5:   if $|B| = N$ then
6:     return
7:   $BoundUncertaintyExplore(B, b^{a^*,o^*}, t+1)$
8:   $\bar{V}(b) \leftarrow J\bar{V}(b)$
```

GapMin uses exactly this tactic — choose beliefs first and update them later. It hence required no modification to fit our framework, except for limiting the number of collected points to N (Algorithm 11).

Algorithm 11 Collection of belief points guided by the bound uncertainty heuristic (GapMin)

```
Function GapMinCollect(N)
1: $B \leftarrow \phi$
2: $pq \leftarrow$ the empty priority queue
3: $pq.Insert(< b_0, \bar{V}(b_0) - \underline{V}(b_0), 1, 0 >)$
4: while $|B| < N$ and $pq \neq \phi$ do
5:   $< b, score, prob, depth > \leftarrow pq.ExtractMaxScore()$
6:   $a^* \leftarrow \text{argmax}_a\, Q_{\bar{V}}(b, a')$
7:   if $\bar{V}(b) - Q_{\bar{V}}(b, a^*) > tolerance$ then
8:     $\bar{V} \leftarrow \bar{V} \cup \{b, Q_{\bar{V}}(b, a^*)\}$
9:   $\alpha \leftarrow backup(\underline{V}, b)$
10:  if $\alpha \cdot b - \underline{V}(b) > tolerance$ then
11:    $B \leftarrow B \cup b$
12:  for each $o \in \Omega$ do
13:    $gap \leftarrow \bar{V}(b_{a^*,o}) - \underline{V}(b_{a^*,o})$
14:    if $\gamma^{depth+1} \cdot gap > tolerance$ then
15:      $prob_o \leftarrow prob \cdot \Pr(o|b, a)$
16:      $score \leftarrow prob_o \cdot \gamma^{depth+1} \cdot gap$
17:      $pq.Insert(< b_{a^*,o}, score, prob_o, depth + 1 >)$
```

The sampling-based L_1 norm belief set expansion collection algorithm, outlined in Algorithm 12, is based on the original PBVI algorithm (Pineau et al, 2003a). Unlike in the original PBVI implementation, where the size of the belief set was doubled at each collection step by creating a successor belief for each $b \in B$, here we only increase the set by N, by randomly picking N parent beliefs from B. This is done again to allow a consistent comparison between methods and control the various experimental conditions.

Algorithm 12 Sampling-based implementation of L_1 norm belief set expansion (PBVI)

```
Function PBVICollect(B,N)
1: Initialize $B_{new} \leftarrow B$
2: while $|B_{new}| - |B| < N$ do
3:    Choose $b \in B$
4:    for each $a \in A$ do
5:       Choose $o \in \Omega$ following the $\Pr(o|b,a)$ distribution
6:       $b'_a \leftarrow b^{a,o}$
7:    $B_{new} \leftarrow B_{new} \cup \text{argmax}_{a \in A} |b'_a - B|_L$
8: return $B_{new}$
```

The Error Minimization collection method (Algorithm 13) is the expansion method of the PEMA algorithm. As with the other methods, we collect N belief points, instead of only a single new belief per iteration. Each new collected belief is considered for being the parent belief point for the next belief.

Algorithm 13 Error Minimization Collection Algorithm (PEMA)

```
Function PEMACollect(B,N)
1: Initialize $B_{new} \leftarrow B$
2: while $|B_{new}| - |B| < N$ do
3:    $b_{max} = \max_{b \in B \cup B_{new}} \bar{\epsilon}(b)$
4:    $b' = \max_{a \in A, o \in \Omega} \Pr(o|b,a)\hat{\epsilon}(b^{a,o})$
5:    $B_{new} \leftarrow B_{new} \cup \{b'\}$
6: return $B_{new}$
```

Table 2 summarizes the computational complexity of each of the belief collection subroutines.

An obvious omission from the set of methods above is the SARSOP algorithm (Kurniawati et al, 2008). This is because in the SARSOP algorithm, the various components (belief collection, value function update, vector pruning) are tightly tied together. After a thorough investigation of various approaches for separating the various components in a way that would not diminish their performance considerably, we have concluded that this cannot be done. Thus, we decided not to include SARSOP in this experimental study.

5.1.2 Value Function Update Methods

Once the beliefs have been collected, point-based solvers need to decide where to apply value function updates. This phase is expressed in the update oper-

Belief Collection Method	Complexity (per belief point)	Requires MDP Solution?
RandomCollect	$O(\|S\|^2 + \|\Omega\|)$	No
FSVICollect	$O(\|S\|^2 + \|\Omega\|)$	Yes
PBVICollect	$O\Big(\|A\| \times (\|S\| \times \|B\| + \|S\|^2 + \|\Omega\|)\Big)$	No
HSVICollect	$O\Big(\|\Omega\| \times \|S\| \times (\|A\| \times \|S\| + \|A\| \times \|\bar{V}\| + \|\underline{V}\|)\Big)$	Yes
GapMinCollect	$O\Big(\|\Omega\| \times \|S\| \times (log(\|B\| + \|A\| \times \|S\| + \|A\| \times \|\bar{V}\| + \|\underline{V}\|)\Big)$	Yes
PEMACollect	$O(\|B\|^2 \times \|S\|^3 \times \|A\| \times \|Z\|)$	No

Table 2 A summary of the computational complexity of belief collection methods, where $|S|$=number of states, $|A|$=number of actions, $|\Omega|$=number of observations, $|B|$=number of belief points, $|\bar{V}|$=number of belief-value mappings in the upper-bound, $|\underline{V}|$=number of α-vectors. The computations above assume that the projection of a belief onto the upper bound requires $|\bar{V}| \times |S|$ computations.

ation of the generic point-based method, Algorithm 2:

$$V_{t+1} \leftarrow UPDATE(V_t, B, B_{new}). \tag{48}$$

There are a number of important considerations at this phase; first we need to decide at which belief points to apply Bellman backup. The order of backups over the selected points is also important.

In a **Full Backup**, we execute *backup*(b) on each belief in the full set of belief points, including the most recent points, i.e. $B \cup B_{new}$. This is the backup method originally used in PBVI and PEMA. Backing up a belief point multiple times may be advantageous when belief collection is particularly expensive (e.g. as in PEMA), or when the values of the successor beliefs are changing. When the computational cost of expanding B is low (e.g., FSVI and Perseus), or when the points collected at first have a lower value (e.g., HSVI collects more important points as the upper bound becomes tighter), a full backup may be wasteful.

In **Newest Points Backup**, the *backup*(b) operator is applied only to the points $b \in B_{new}$. This approach was first suggested as part of trajectory-based algorithms, such as HSVI. These algorithms construct at each belief expansion phase a trial that begins at b_0. As such, improvements to the policy would be made across the entire planning horizon. In methods such as PBVI, where the belief tree is expanded by adding new leaves, this method makes little sense, because if we update only the values at the newly added leaves, the change in value will not be noticed at the inner nodes.

An alternative to these two methods was suggested in the Perseus algorithm (Spaan and Vlassis, 2005). This approach, henceforth called the **Perseus-Backup**, uses a random order of backups, but ensures that after each iteration the values of all beliefs have been improved. Intuitively, this method encourages smaller value functions, but may be problematic in domains where the order of backups is important.

5.1.3 Value Function Pruning

As we have discussed above, pruning is less crucial in point-based methods than in exact solvers, but some simple pruning can still be useful in reducing computation time and memory consumption.

During belief updating, for each belief point b, we compute $\alpha_{new} = backup(b)$. However, instead of directly adding α_{new} to V, we check whether the new vector has improved or maintained the value at the current point b — $\alpha_{new} \cdot b > V(b)$. We discard vectors that do not satisfy this condition. Note that in doing so, it is possible that we will prune α-vectors which would be optimal for some other $b' \neq b$. Still, this optimization provides a considerable reduction in the size of V and hence a significant speedup in practice.

In the α-vector based value function representation we care only about the upper envelope of the value function. Vectors that do not participate in this upper envelope are called dominated. One could prune dominated vectors using a procedure that employs linear programs, but this procedure is computationally expensive. However, some vectors may be completely dominated by a single vector. That is, there may exist an $\alpha \in V$ such that; $\alpha(s) \leq \alpha_{new}(s)$ for all $s \in S$. Such vectors are called vectors pointwise dominated. When adding α_{new} to V, we prune all vectors that are pointwise dominated by α_{new}. Note that α_{new} cannot be dominated because we already insured that $\alpha_{new} \cdot b > V(b)$.

Some methods, in particular SARSOP, include other routines for pruning the set of α-vectors. We did not investigate this issue in our empirical work, because successful pruning of V is still in many ways an under-explored issue, especially with respect to its impact on balancing time and memory, and we decided to focus on other important questions in this paper.

5.2 Domains

In recent years, the POMDP community has evolved a set of benchmarks problems, each motivated by an interesting real-world problem, such as path planning, human-robot interaction, space exploration, or knowledge discovery. We hand-picked a number of domains from this benchmark set, with the goal of spanning the range of properties that affect the performance of point-based algorithms. Below, we review the selected domains and their properties.

Domain	$\|S\|$	$\|A\|$	$\|\Omega\|$	Transitions	Observations	Horizon
Hallway2	92	5	17	stochastic	stochastic	29
Tag	870	5	30	stochastic	deterministic	30
RockSample[7,8]	12545	13	2	deterministic	stochastic	33
Underwater Navigation	2653	6	102	deterministic	deterministic	62
Tiger-grid	36	5	17	stochastic	stochastic	8
Wumpus 7	1568	4	2	deterministic	deterministic	16

Table 3 The domains used in our experiments. Horizon is a crude estimation of the average number of steps required to solve the problem, i.e. reach a goal or a reward state.

Hallway2 (Littman et al, 1995) is a classic small robot navigation domain. There are several other similar benchmark problems that were proposed around the same time. These domains are relatively small compared to newer benchmarks, in terms of the number of states and observations, but exhibit a stochastic transition and observation functions that result in a very dense belief state. That is, throughout the execution of a policy, many states have non-zero probabilities.

Tiger-grid (Littman et al, 1995) is another classic example, designed to demonstrate the value of information. An agent must figure out its location in a maze in order to find the right exit. The agent must take actions that move it outside the path to the door in order to discover where it is. While this domain is relatively small, it can be surprisingly challenging, even for modern solvers due to high state aliasing.

In both Hallway2 and Tiger-grid, we used a slightly different version of the benchmark, where the goal state is an absorbing, zero-reward, sink state. Due to historical settings (Littman et al, 1995), experiments on these benchmarks were stopped once the goal state has been reached, and reported ADRs on these domains in the community used this non-standard setting, while the computed value function using the model definition considered restarts once the goal state has been reached. Replacing restarts with absorbing states removed the need to stop the ADR computation once the goal has been reached.

Tag (Pineau et al, 2003a) (also called sometimes TagAvoid) is a domain where an agent tries to find an opponent in a T-shaped room. The opponent is purposefully hiding, and the agent observes it only if it is in the same location as the opponent. The agent movements are stochastic and it has full observability over its own location. The opponent moves stochastically away from the agent. This domain scales to larger state spaces than the traditional maze domains, and has been used as a benchmark for most methods included in our survey. The deterministic observations cause only a small subset of the states (less than 30) to have a non-zero belief. Furthermore, the opponent is typically driven to one of the three corners of the T-shaped room, and thus is rather easy to find.

RockSample is a scalable navigation problem that models rover exploration (Smith and Simmons, 2004). An instance of RockSample with a map size $n \times n$ and k rocks is denoted as RockSample$[n, k]$. The agent needs to sample a subset of the rocks that are interesting, and has a long range sensor to establish whether a rock is interesting or not, with a decreasing accuracy as the agent gets farther from the inspected rock. RockSample is an interesting domain to include since it is not strictly goal oriented.

The Underwater Navigation domain (Kurniawati et al, 2008) is an instance of a coastal navigation problem. This domain has a large state and observation spaces. However, it is simple in the sense that the transitions and observations are deterministic. A primary difficulty in the Underwater Navigation domain is that there is substantial aliasing since the state space is much larger than the observation space. The required planning horizon is also relatively long.

Wumpus is a domain motivated by problems of the same name from the planning community (Albore et al, 2009). An agent navigates through a grid that hosts several monsters, called Wumpuses, along the path to the goal. When the agent is near a cell hosting a Wumpus, it can smell the Wumpus. However, it does not know where the smell comes from, i.e., which of the neighboring cells contains a Wumpus. The agent must visit several neighboring cells in order to reason about the Wumpus whereabouts. This domain is challenging because relatively lengthy detours must be made in order to know whether a cell is safe or not.

5.3 Metrics

In POMDP planning, the goal is typically to optimize the expected stream of discounted rewards over the planning horizon when following a policy π:

$$E\left[\sum_{t=0}^{T}\gamma^t r_t|\pi\right]. \tag{49}$$

Computing this expectation exactly requires us to examine all possible action-observation trajectories of length T following the policy π. As there are $|\Omega|$ possible observations following each action selected by π, the number of such trajectories grows exponentially with the horizon T. Thus, this expectation cannot be computed exactly in general. We can, however, estimate the expectation by simulating sampled trajectories (trials) in the environment. In each trial, the agent begins at b_0, and executes actions following π. We average over multiple executions to obtain an unbiased estimator of the expectation, known as the Average Discounted Reward (ADR):

$$ADR = \frac{1}{n}\sum_{i=1}^{n}\sum_{t=0}^{T}\gamma^t r_t|\pi, \tag{50}$$

where n is the number of trials, and T is an upper bound on the trial length. For domains with infinite planning horizons, as $\sum_{t=0}^{\infty}\gamma^t R_{max} = \frac{R_{max}}{1-\gamma}$, the difference between stopping at time T and continuing to infinity is at most $\frac{\gamma^T(R_{max}-R_{min})}{1-\gamma}$. Hence, we can set T such that this difference is bounded by a predefined ϵ.

In measuring the ADR, there is variance due to both the stochastic nature of the domains, and, for some algorithms, due to stochasticity in the solver itself. To control for these aspects, we execute each algorithm over the same domain 50 times, and average the results at each time point. For clarity, we omit the confidence intervals in most of our results (except for Section 6.2).

Throughout our results, we present the quality of the resulting policies given a range of different planning times. This approach not only lets us detect when a given algorithm has converged, but also allows us to evaluate the rate of convergence, to understand which methods approach a good solution more quickly. The ADR is reported at specific (pre-selected and evenly distributed) time points. To enforce this, we report the ADR for the best policy produced (full iteration of belief collection and value update completed) before t; we do not extrapolate between solutions.

When presenting the results, we compare the ADR achieved by various methods over wall-clock time. Wall-clock time is problematic when comparing results over different machine architectures and implementations. It is arguably better to report other measures such as the number of algorithmic steps, which generalize across machines. However, in our case, agreeing on what constitutes a "step" is difficult given the different approaches that we compare. Furthermore, all our algorithms use the same infrastructure, i.e., belief update and point-based backup operations. Therefore, we believe that comparing wall-clock time is a reasonable approach for the purposes of this analysis.

6 Empirical Analysis of Point-Based Methods

All the algorithms were implemented over the same framework, using identical basic operations, such as belief update and point-based backups. We use sparse representations for beliefs and non-sparse representations for α-vectors, where typically there is a large variation in the different entries. The implementation is highly optimized for all algorithms. All experiments were run on a 2.66Ghz Xeon(R) CPU with 24 cores (although the implementation is not multi-threaded) and 32GB of system memory. Results were computed by averaging over 50 different POMDP solutions, since many solvers have stochastic components. The averaged discounted return (ADR) was computed by simulating the policy on the environment for 500 trajectories (for each of the 50 different solutions). The ϵ parameter for HSVICollect was set to $\epsilon = 0.001$. Empirically, the results are not very sensitive to this parameter.

6.1 Choosing a method for belief collection

The first set of experiments explores the performance of various belief collection routines. The objective is to see if we can draw overall conclusions on how the choice of collection method affects the speed of convergence, as well as seeing whether this is domain-dependent. Throughout these experiments, we vary the belief selection routine, but keep other parameters mostly fixed. Unless otherwise specified, the number of points collected at each step is $N = 100$ for all collection methods, except PEMACollect which uses $N = 10$ (since this collection routine is much slower than the others). The value function is updated using a FullBackup, and unless mentioned otherwise, we do a single round of value updates per iteration ($U = 1$).

The results for the Hallway2 and TagAvoid domains are presented in the top row of Figure 1. The primary observation is that many methods find a solution within 50 seconds. We observe that HSVICollect and GapMinCollect

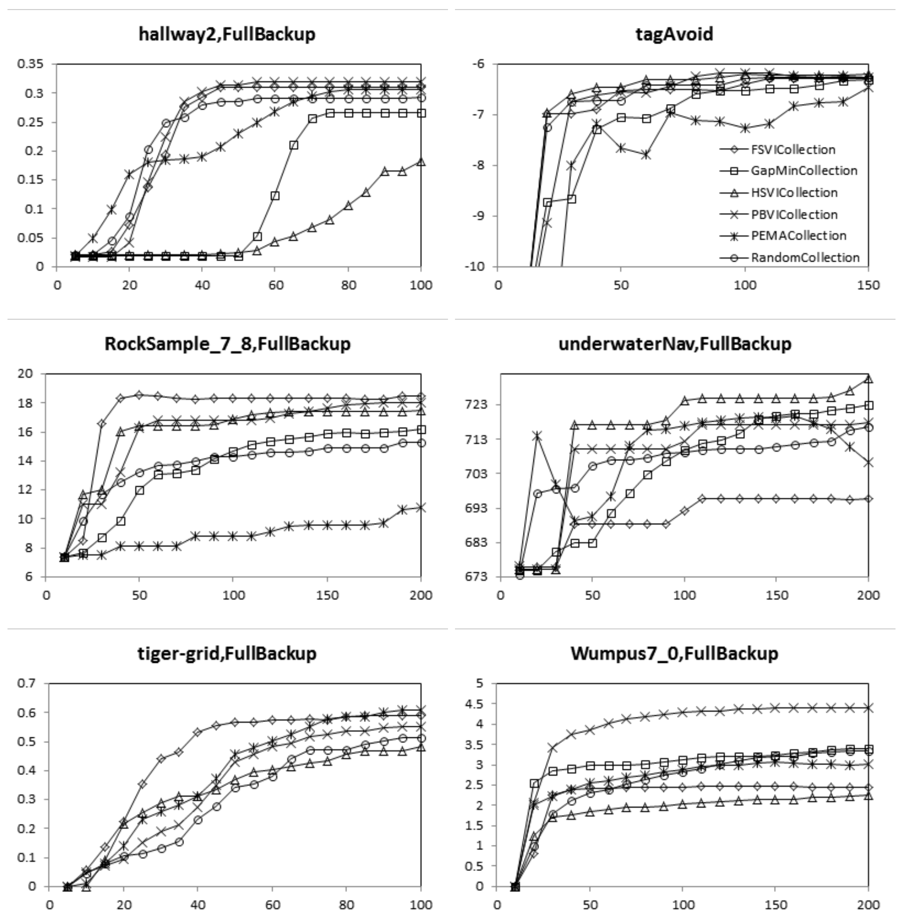

Fig. 1 Comparison of different belief collection methods in six contrasting domains. Value updates are done using the FullBackup method. We collect $N = 100$ new belief points ($N = 10$ for PEMACollect) at each iteration. We perform $U = 1$ value update over all points. All graphs show ADR in the Y axis given time (seconds) in the X axis.

are slower for Hallway2. This is attributed to the fact that this domain features many observations, which causes a high branching factor in the belief search. GapMinCollect starts slowly, but then rapidly converges to a good solution, probably once the breadth-first search found a belief with a high probability on the goal state. PEMACollect is slower in TagAvoid. As we will see later, it does not perform very well in many domains, mainly due to scalability issues in domains with large number of states (more than a few dozen).

Next, we consider two types of domains — navigation-type domains, and information-type domains; two examples of each are presented. These are more challenging domains than those presented above.

The middle row of Figure 1 shows the results for the two navigation-type domains, known in the literature as RockSample[7,8] and UnderwaterNavigation. Here we see more differentiation between belief collection methods. These domains are larger than the ones considered in the top row, and require more informed belief collection. FSVICollect is the best method for the RockSample domain. The MDP heuristic leads it rapidly to the good rocks, and in this domain there is no need for moving away from rewards to collect information. RandomCollect does poorly in this domain, because there are many random trajectories here that do not lead to the interesting rewards. GapMinCollect also does not do well on this domain while HSVICollect does fairly well, probably because with the relatively long planning horizon in this domain is more appropriate for depth first, rather than breadth first search. PEMACollect is well below all other methods, due to the significant computation time of each belief expansion (cubic in the number of states).

We see more differentiation between methods in UnderwaterNavigation. Here, HSVICollect performs the best, benefitting from the optimistic exploration bias and the deterministic transitions. GapMinCollect that uses the same action selection heuristic, but explores by breadth first search takes longer but achieves similar performance. PBVICollect and RandomCollect are less good, because they explore many suboptimal branches. FSVICollect also has suboptimal performance here because it does not execute smart information gathering required in this domain. PEMACollect performs very erratically here.

This non-monotonic performance of PEMACollect in UnderwaterNavigation may seem unintuitive; it is not impossible for an algorithm to exhibit such behavior especially when there are few belief points supporting the value function. Recall that PEMACollect adds a very small number of belief points (10, rather than 100) between value update steps. As a result, the solution at time $T = 100$ for PEMACollect is computed with roughly 200 belief points, compared to 500 belief points for HSVI; this is for a domain with 2653 states.

Notice that after 100 seconds of computation in UnderwaterNavigation, most methods have not yet converged. Recall that most of the belief collection methods considered are guaranteed to eventually converge to the optimal solution (FSVICollect being the exception), though may require longer to fully refine their solution. We terminated the experiments after 200 seconds to better illustrate the differences in convergence speeds between the different collection methods.

Finally, we consider the results for the Tiger-grid and Wumpus domains in the bottom row of Figure 1. Both of these domains require explicit information-gathering sequences to achieve good performance. In the case of Tiger-grid, we observe good performance by FSVICollect, PBVICollect, and PEMACollect. We observe that HSVICollect does not do well in this domain, and GapMinCollect could not compute a reasonable policy in this domain, and remained with an ADR of 0, and was hence removed from the graph.

Tiger-grid is especially bad for GapMin, because in this domain the agent has to collect a differentiating observation before collecting the reward. The gap between bounds is hence maximized when the agent reaches the reward collection *without* previously observing the differentiating observation, because the upper bound predicts the positive reward while the lower bound predicts the negative punishment. The bound is tighter for the beliefs over states where the differentiating observation can be collected, and these are thus never considered. Also, separating the upper bound update and the belief collection, as is naturally done in GapMin, but not in the original HSVI (Smith and Simmons, 2005), is especially damaging here in obtaining good belief points. Note that our results on TigerGrid differ from previously published results, because we removed restarts.

The Wumpus domain is an interesting case for POMDP solvers. It is a domain where widely exploring the belief space — as does PBVICollect — is profitable, whereas tightly guided exploration following an optimistic MDP-driven heuristic, as in GapMinCollect, HSVICollect and FSVICollect, is a significant disadvantage. This advantage towards breadth first exploration is also evident from the superior performance of GapMinCollect compared to HSVICollect in this domain.

6.2 Variance in performance

In all previous graphs, we omitted measures of uncertainty for clarity. Here we show in Figure 2 the standard error over the empirical ADR for different collection methods in the Tiger-grid, Wumpus, and UnderwaterNavigation domains. There are two sources of variance, one from the empirical estimate of the ADR, and the other from the stochasticity in the belief selection mechanisms. In general, the variance is lowered as the algorithms converge towards a good solution. Measures of empirical uncertainty will also be omitted from subsequent graphs to make the graphs more readable; recall that all results presented are computed by averaging over 50 different solutions and 500 trajectories for each.

6.3 Computation profile

The various collection methods considered vary significantly in terms of computational load. Time-dependent results for each method are provided above, yet it is helpful to also observe how this load is distributed between algorithmic components. Figure 3 shows the ratio of CPU time dedicated to the belief collection versus the value updating for each of the different belief collection methods considered. As expected, PEMACollect is by far the most expensive method to acquire beliefs; these results continue to assume $N = 100$ for all methods, except $N = 10$ for PEMACollect. The RandomCollect method generally uses negligible time. FSVICollect is always very fast. PBVICollect is also reasonably fast, though can be slower in domains with more actions. HSVICollect usually requires more computation for belief collection in domains where

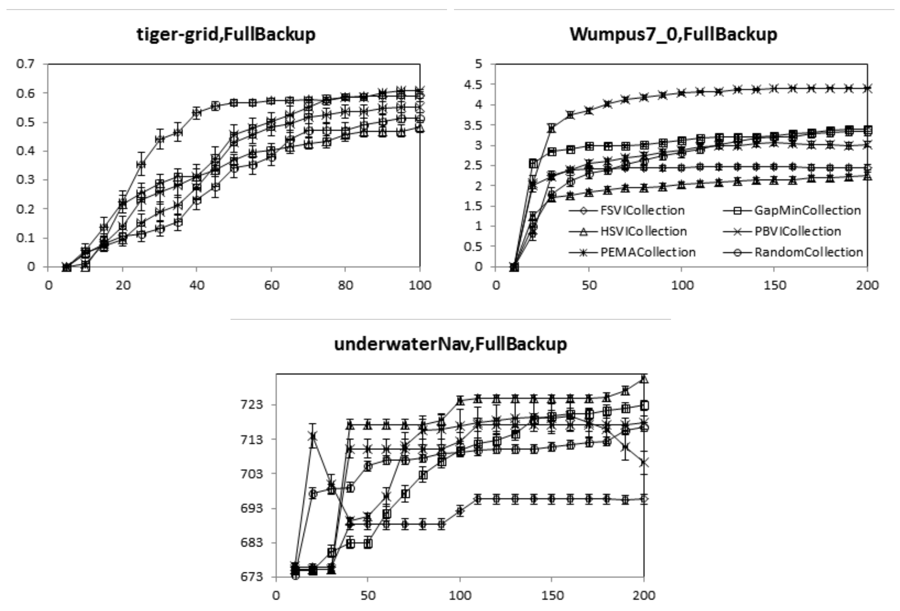

Fig. 2 Standard error over the ADR in the Tiger-grid, Wumpus, and UnderwaterNavigation domains. All collection methods are implemented by repeatedly alternating adding $N = 100$ belief points ($N = 10$ for PEMACollect) with a single value update at all points (FullUpdate). All graphs show ADR in the Y axis given time (seconds) in the X axis.

there is more stochasticity (e.g. Hallway2, Tiger-grid), but less in domains that are deterministic (e.g. UnderwaterNavigation). The results presented are representative of all domains considered.

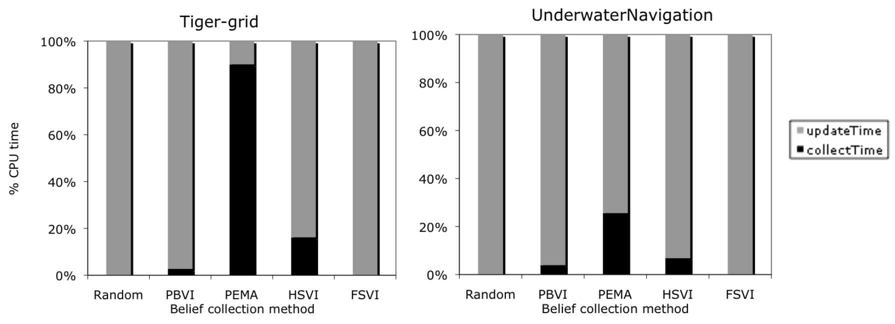

Fig. 3 Distribution of computational load.

6.4 Choosing a schedule for value updates

Next, we examine how the ordering of value updates affects the performance of point-based solvers. Recall that we consider three different strategies: FullBackup, PerseusBackup, NewestPointsBackup.

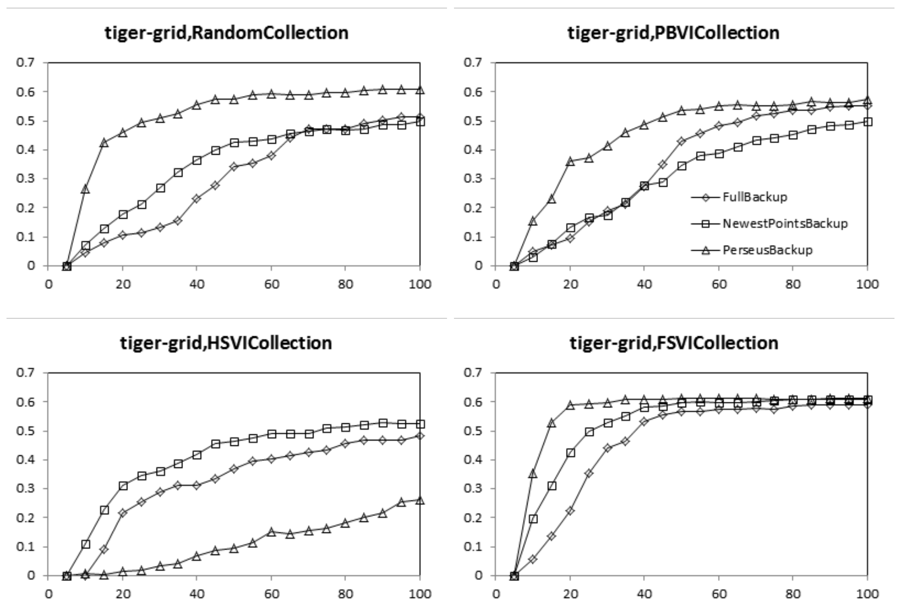

Fig. 4 Comparison of different value function update orderings in the Tiger-grid domain. All graphs show ADR in the Y axis given time (seconds) in the X axis.

In Hallway2 (graphs omitted) , the choice of backup ordering did not matter for the collection methods that performed well (PBVICollect, FSVICollect and RandomCollect). For the other two (HSVICollect and RandomCollect), we observed a negative effect of switching to either PerseusBackup or NewestPointsBackup. In Tag (graphs omitted) and RockSample (graphs omitted), we observed very little effect in terms of the choice of backup ordering approach. Only when considering PEMACollect did we observe a negative effect to using PerseusBackup or NewestPointsBackup (see below, Figure 7).

In Tiger-grid (Figure 4), the choice of backup ordering was less important when using FSVICollect with FullBackup being somewhat worse. When using RandomCollect, the PerseusBackup, which is the combination suggested originally in the Perseus algorithm (Spaan and Vlassis, 2005) performed the best. PerseusBackup also works better with PBVICollect, although this is less pronounced. When using HSVICollect, which is the less preferred collection method for TigerGrid, NewestPointsBackup, the original combination of the HSVI algorithm, is better.

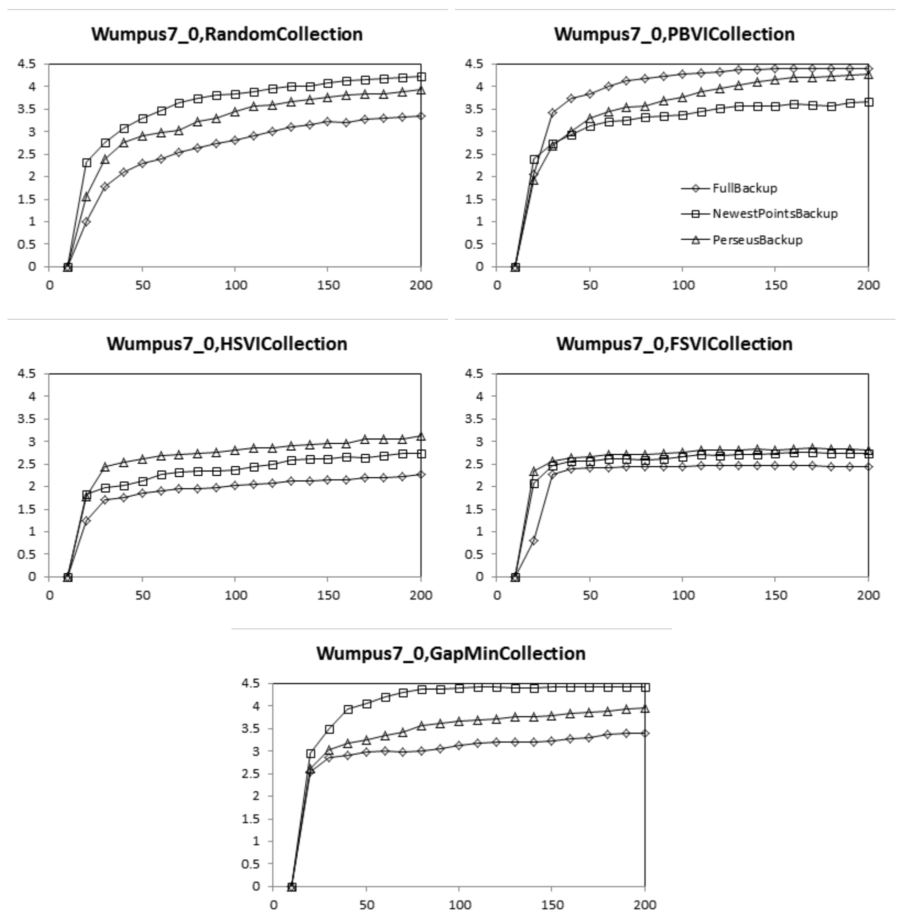

Fig. 5 Comparison of different value function update orderings in the Wumpus domain. All graphs show ADR in the Y axis given time (seconds) in the X axis.

In the Wumpus domain, we observe a different effect. As shown in Figure 5, NewestPointsBackup is the top ordering method for two of the leading collection methods — RandomCollect and GapMinCollect. This difference is most evident when using GapMinCollect and NewestPointsBackup together, the original combination of the GapMin algorithm, that causes GapMinCollect to perform better than all other combinations. The FullBackup tends to perform poorly, except when combined with PBVICollect, where it is the best combination. This is probably because the rest of the methods collect many redundant points, and updating all these points repeatedly reduces performance. This conclusion is farther supported by observing the effect of removing duplicate belief points from the set B in this domain, as we show later in Section 6.8. With all ordering methods, both HSVICollect and FSVICollect perform well below the (presumed) optimal solution obtained when using

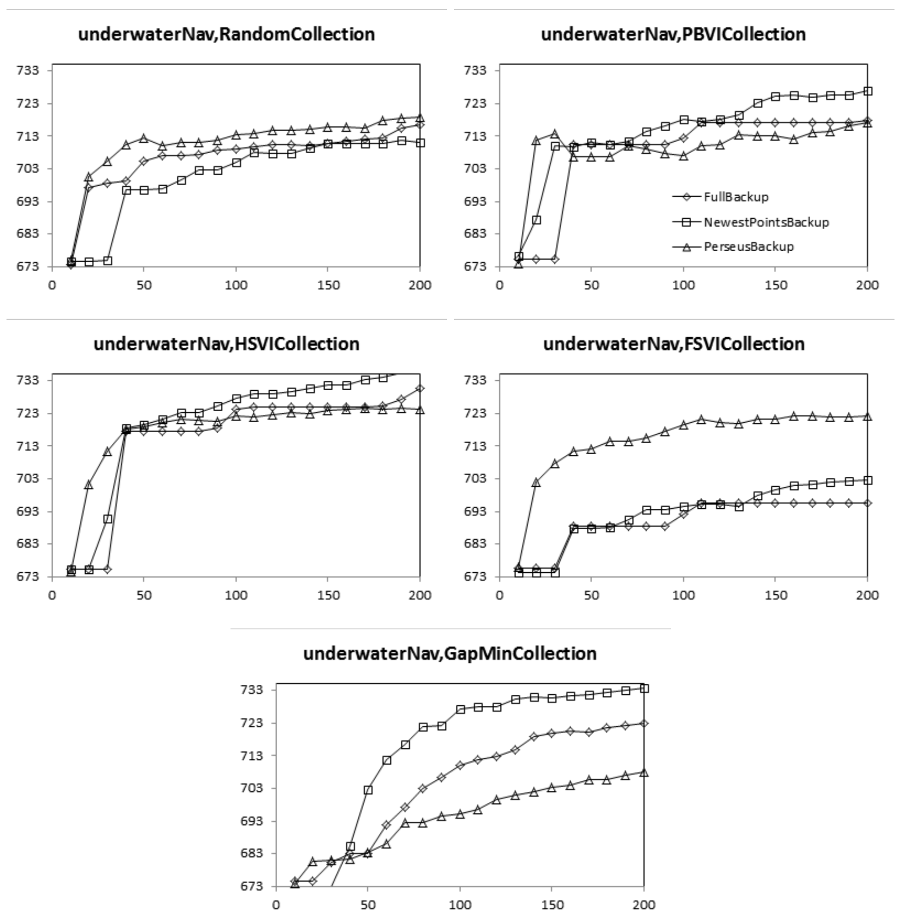

Fig. 6 Comparison of different value function update orderings in the UnderwaterNavigation domain. All graphs show ADR in the Y axis given time (seconds) in the X axis.

PBVICollect, GapMinCollect, or RandomCollect, which benefit from a wider, less goal-oriented, exploration of the belief space.

In the UnderwaterNavigation domain, the best update method is highly dependent on the choice of collection method. As shown in Figure 6, the PerseusBackup is best for RandomCollect and even more significantly so for FSVICollect. In fact, with PerseusBackup FSVICollect becomes almost competitive for this domain. For PBVICollect, HSVICollect, and GapMinCollect, the NewestUpdate performs best. Note that in this domain, HSVICollect achieves the best policies, with GapMinCollect trailing behind. As with Wumpus, GapMinCollect is the most sensitive to the choice of the backup ordering method.

Finally, throughout all of our experiments with PEMACollect, it was always better to apply a FullBackup. Other methods can give good results, but less consistently. This is not surprising since PEMACollect is very careful

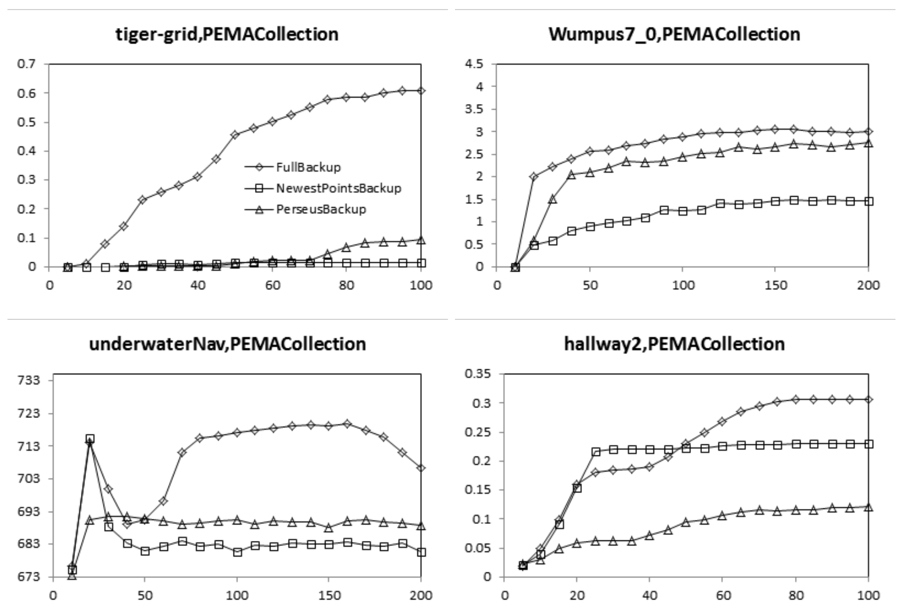

Fig. 7 Comparison of different value function update orderings in combination with PEMA-Collect. All graphs show ADR in the Y axis given time (seconds) in the X axis.

about which points it selects, thus it stands to reason that it pays off to update all of them. Figure 7 shows this effect for a few domains, but similar results were observed on all domains. The effect was particularly noticeable for Tiger-grid. Of course this primarily applies to reasonably small domains, otherwise PEMACollect is not a good option, regardless of value updating schedule.

Observing the results above, we can conclude best matches for algorithmic components over various domains, presented in Table 4. We can immediately see that NewestPointsBackup is preferred in most cases, and FullBackup is preferred only by PBVICollect, and even then not in all cases. In the coming sections, when comparing other features of the algorithms, we will use only these best combinations.

Domain	Random	PBVI	HSVI	GapMin	FSVI
Wumpus 7	Newest	Full	Perseus	Newest	Newest
underwaterNav	Perseus	Newest	Newest	Newest	Perseus
Tiger-grid	Perseus	Perseus	Newest	×	Perseus
RockSample [7,8]	Newest	Full	Newest	Newest	Newest

Table 4 the best combinations for the domains that will be investigated in Section 6.5 and on.

6.5 Choosing how many belief points to add

The results presented above in Figure 3 show how different methods balance computation load between exploring the belief space and updating value estimates. In general, the balance between these two components can be controlled by choosing the number of belief points collected at each iteration (N), and the number of iterations of belief updates (U) performed between adding batches of points. When N is high, this shifts the balance of computation time towards belief point collection, especially when the time complexity of the update method does not scale linearly with N (as in the Perseus update method).

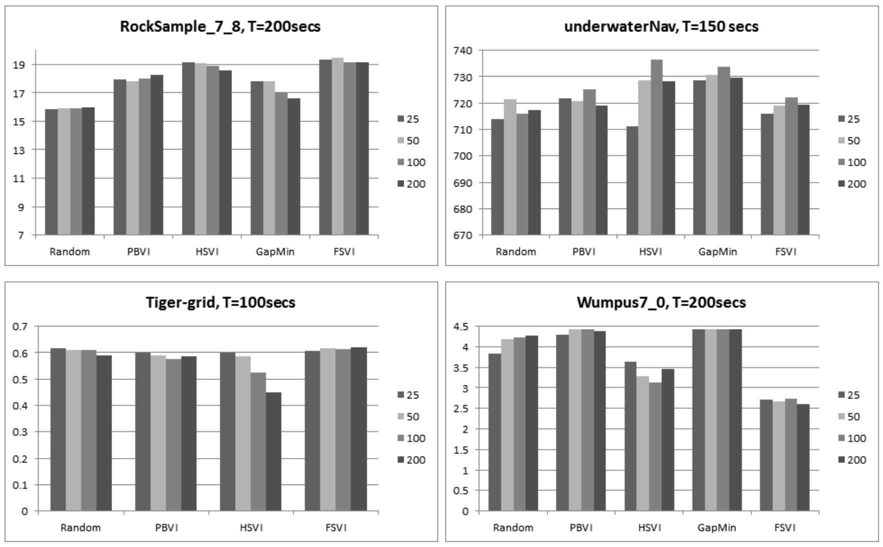

Fig. 8 Comparison of different numbers of belief points to add between iterations of value updates. We vary the update schedule for each domain, according to the results of Section 6.4. We show the average discounted reward (ADR) after the planning time specified for each domain.

Modifying U has the opposite effect: when we increase U, we linearly increase the time spent computing belief point updates while the collection time remains fixed. This should theoretically be useful for collection/update combinations that spend a large proportion of time collecting, compared to updating. We begin by exploring the effect of the number of belief points in this section; the issue of the number of belief updates is explored in the following section.

For the four domains presented in Figure 8 we consider $N = \{25, 50, 100, 200\}$. Recall that $N = 100$ was used for all results presented thus far. For the most part, we observe that empirical results are relatively stable with respect to the parameter N. HSVICollect is perhaps the most sensitive to this parameter in three of the domains. In the UnderwaterNavigation domain, for example, with

HSVICollect, we do notice a notable improvement with larger number of belief points (from 25 to 100), and then a decrease from 100 to 200 points, where too many belief points are collected, and more time is needed for updating the value function. UnderwaterNavigation is the most sensitive in general to this parameter, probably due to the relatively long planning horizon. Other domains and methods see a small improvement or decrease in performance as a function of the number of beliefs added, but the effect is generally modest. This suggests that it is not too important to carefully tune this parameter, which is useful to remember when tackling a completely new domain.

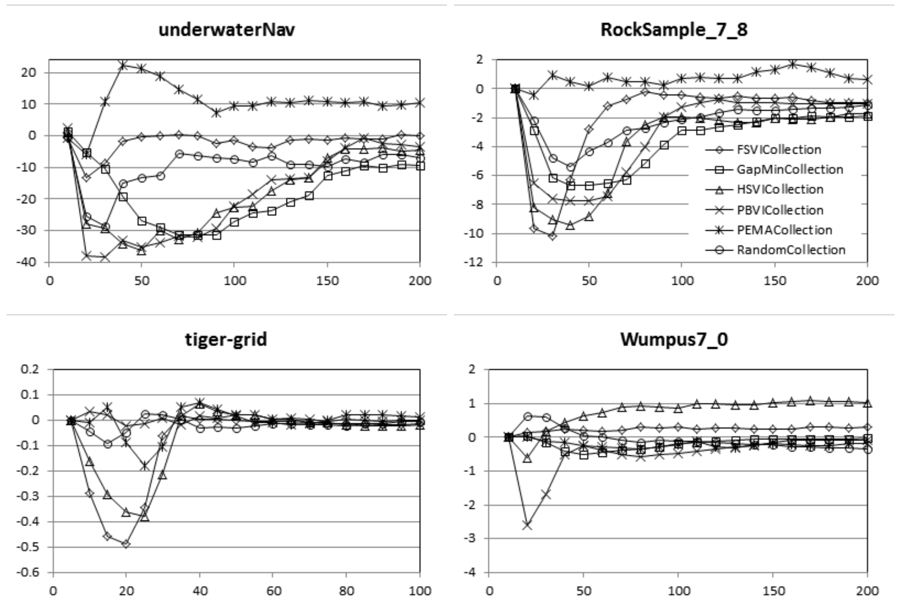

Fig. 9 Comparison of different numbers of value function updates between rounds of adding new belief points. For each belief collection method, we compare running 5 value updates for all belief points, compared to only 1 value update. The y-axis represents the difference in ADR when using using 5 value updates instead of one. When using 5 updates is useful, the value should be higher than 0.

6.6 Choosing how many value updates to perform

In this section, we vary the parameter U, which controls the number of iterations of belief point updates at each step in the POMDP planning. Applying more updates may result in a better policy than a single update, however the extra time spent updating is not used for collecting new points (and updating these points instead).

In fact, as shown in Figure 9, this is exactly what happens; performance usually degrades or stays the same with the addition of more value backups. The only exception is with PEMACollect, whose performance improves with more updates, but is still much below that of other methods. This is not unexpected since with such a slow collection algorithm, it is useful to extract as good a policy as possible with the belief points that have been collected. Results are similar across domains.

6.7 Belief ordering

One of the interesting innovations proposed in the HSVI paper was to update the value at belief points in the reverse order in which the points were acquired. Assuming for example that we collected points $b_0 ... b_T$, then when executing a round of updating, we would update the value at each belief starting with b_T, and moving back. This procedure is used in much of the reinforcement learning literature, where the idea has been extended in various ways, for example using eligibility traces (Singh and Sutton, 1996). We expect this ordering to matter most for domains with long planning sequences, such as UnderwaterNavigation, since it provides a way to pass the value updates backward from the goal to earlier beliefs.

In Figure 10, we show the difference in ADR when the beliefs are updated in reversed order instead of the same order as they are collected. Rather surprisingly, the effect is not very large, and in some cases, the forward ordering performs just as well, or slightly better. As expected, the benefit from reverse ordering is more pronounced in the Wumpus, RockSample, and the UnderwaterNavigation domains, that have longer planning horizons, and less so in the TigerGrid domain.

It should be emphasized that all other results presented in this paper (except, of course, some of the results of the current section) use the reverse ordering.

6.8 Removing duplicate beliefs

As mentioned above, we did not investigate in detail the issue of pruning (beliefs or α-vectors), since it is a substantial topic, requiring a discussion on its own. However we performed a simple experiment, whereby we considered removing any duplicates from the belief set. This can be computed quickly, and easily. Some of the previous literature on Perseus had suggested that including duplicate beliefs may be beneficial. Duplicate beliefs occur only with HSVICollect, FSVICollect and RandomCollect; the collection criteria in GapMinCollect, PBVICollect, and PEMACollect are such that duplicate beliefs are never selected.

We see from the results shown in Figure 11 that in some domains, in particular those with information-gathering aspects such as Wumpus, it can

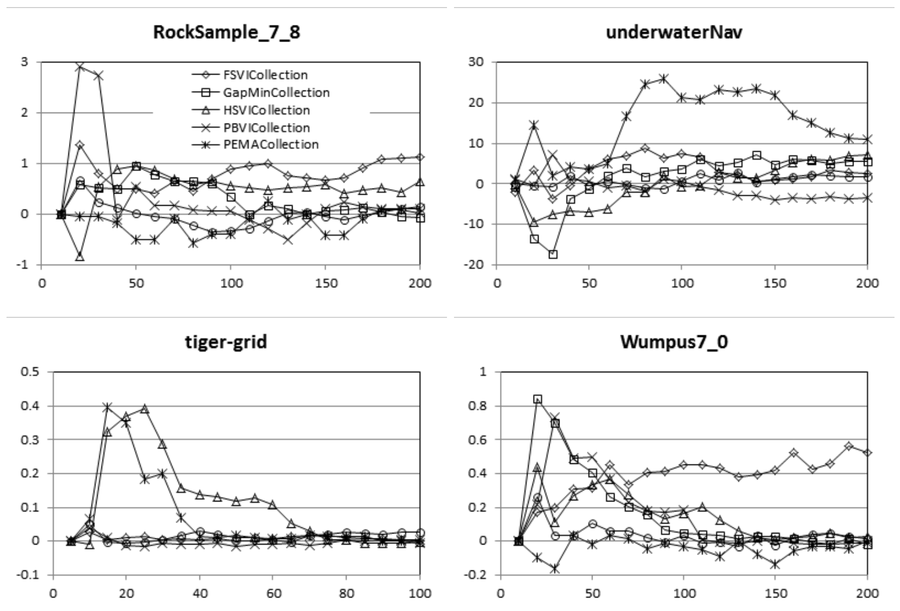

Fig. 10 Comparing forward and reverse ordering of collected belief points while updating the value function. The y-axis represents the difference in ADR when using ordering the beliefs in reverse order instead of by the order by which they were collected. When reverse ordering is useful, the value should be higher than 0.

be beneficial to remove duplicates, since it allows increased exploration of the belief space. In other more goal-directed domains, such as RockSample, the effect is modest to non-existent (within 0.5 of an ADR of around 18).

6.9 Initializing the value function

Another design decision that arises when implementing a point-based solver is the choice of initial value function, V_0. As discussed in Section 3.5, some methods require the initial function to be a lower bound on the optimal, V^*, whereas others perform well regardless of the initial value function. In the updated version of HSVI (Smith and Simmons, 2005), the initial lower bound is based on a blind policy, as first proposed by Hauskrecht (1997). The idea is to create an initial value function that contains one α-vector per action, where each α-vector represents the policy of always taking the associated action (and only that action).

This blind lower bound was used in all experiments reported above. This bound is guaranteed to be tighter than a single vector lower bound initialized according to the minimum reward, yet a naive lower bound (defined in Equation 38) is somewhat easier to program. In Figure 12, we show the impact of this choice, by plotting the difference between the empirical return obtained

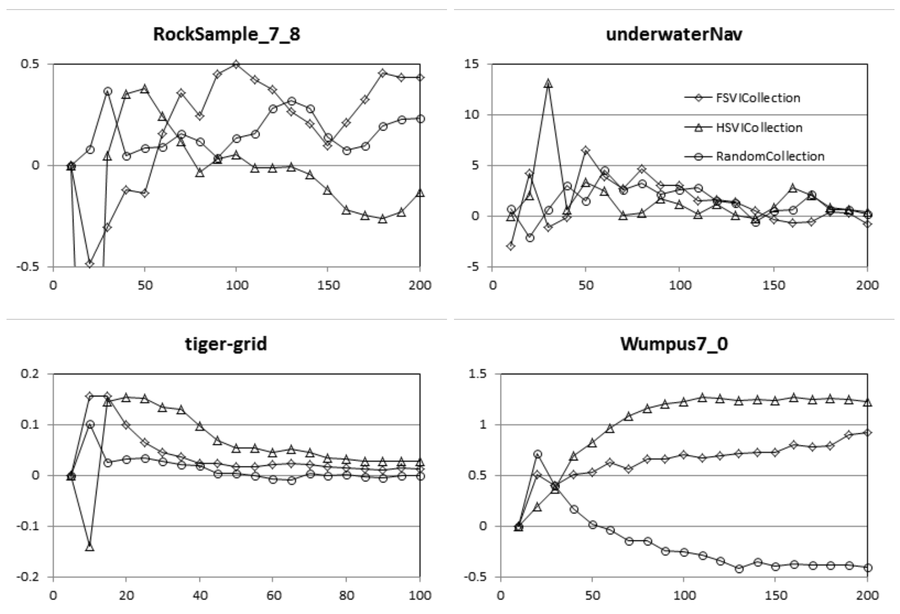

Fig. 11 Measuring the effect of removing duplicates in the belief set, compared to allowing duplicates. The y-axis represents the difference in ADR when using removing duplicate beliefs instead of allowing duplicate beliefs. When removing duplicates is useful, the value should be higher than 0.

when using a Blind lower bound, and the empirical return obtained when using a naive lower bound to initialize the value.

Overall, we observe that the disadvantage in the extra computation required by the blind lower bound is minimal, and the impact on performance can be significant, especially for short planning times.

In the UnderwaterNavigation domain, there is a significant degradation of performance when initializing with the naive lower bound. This is because in this domain there are very large punishments (negative rewards) that are easily avoided. The naive lower bound hence reflects receiving these punishments forever, while the blind strategy avoids them. Hence, many backups are needed when initializing using the naive lower bound before obtaining a reasonable policy. In fact, PBVICollect and PEMACollect never overcame the initialization and hence the difference for these two methods is considerable.

It is also interesting to note that GapMinCollect, while benefiting at the beginning from the tighter initialization, manages later to compute a similar value function using the two initialization methods. As GapMinCollect is one of the best methods in this domain, this points to a strength of the GapMin strategy in overcoming a bad initialization.

In other domains a naive initialization of the value function is quickly overcome and after a few seconds of planning, the performance is nearly identical for both types of value function initialization. This may reflect the fact that this is an easier planning domain.

Finally, in some cases the naive lower bound leads to slightly better performance (e.g. FSVICollect in the Wumpus domain), which is interesting. It may be that using a less constrained lower bound allows the value iteration to construct a better shaped value function, and hence get some benefit. That being said, FSVICollect and PEMACollect do not perform well on Wumpus, so even with the naive lower bound they are far from being the best methods for this domain.

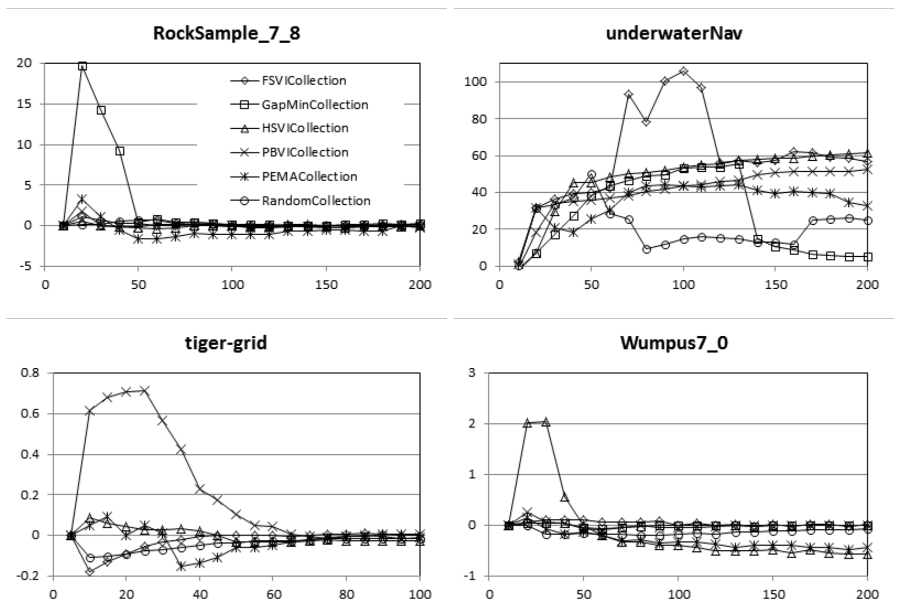

Fig. 12 Measuring the effect of value function initialization, by plotting the difference between the ADR obtained when using a Blind lower bound and the ADR obtained when using a naive lower bound. The y-axis represents the difference in ADR when using the Blind lower bound, compared to the naive lower bound. When the Blind lower is useful, the value should be higher than 0.

7 Discussion

In this section we present the conclusions of our analysis, and suggest directions for future investigation.

7.1 Summary of empirical analysis

We begin by summarizing a number of key findings from the empirical analysis.

– We observe that different methods are suited for different types of environments. Thus, when solving a new problem, it is important to consider the characteristics of one's domains when picking a solver, rather than just picking the latest method.

– The size of the problem domain is not a direct indicator of the difficult of finding a solution. Some problems with a large state space can have a very simple structure that makes them easier to solve. For example, the TagAvoid domain has substantially more states than the Tiger-grid domain, yet the latter takes longer to solve. In this case, it is due to the fact that one of the factors of the state space (the robot's position) is fully observable, limiting the partial observability to the other factor (the target's position). In general, the number of reachable beliefs may be a better indicator of solution difficulty.

– HSVICollect is effective in domains with low branching factors, such as RockSample and UnderwaterNavigation. This is typical of domains with deterministic actions. It is less advantageous in domains with more stochasticity, such as Hallway2, and Tiger-grid and domains with many observations, such as Wumpus, where the branching factor in the belief tree is too large. We expect HSVICollect to be useful in robot planning domains, where deterministic planning algorithms have until recently been fairly useful.

– PBVICollect performs well in domains requiring wide, undirected, exploration of the belief. In a domain like Wumpus, where there is intrinsic value to exploration, it achieves a good solution substantially faster than other methods. We expect it to be useful in human-robot interaction domains, where its exploratory behavior may compensate for high stochasticity and poor domain parameterization.

– FSVICollect performs very well in a domain like Tiger-grid. This is because the stochastic transitions occasionally lead the agent to states where it receives the needed observations. In domains with highly stochastic transitions the error in transition often serves as an exploration factor.

– GapMinCollect (combined with the NewestPointsBackup) has shown good results in some of the larger and more difficult domains, such as Wumpus, and UnderwaterNavigation. In UnderwaterNavigation it was also the only method to overcome a bad initialization, thus showing robustness to initialization techniques. On the other hand, some domains, such as Tiger-grid without restarts are very difficult for GapMinCollect, where it tends to collect beliefs with large gap but ignores beliefs where required observations are collected.

– RandomCollect performs surprisingly well (given how naively it picks points) across domains, though is better suited to highly explorative domains, such as Wumpus and Tiger-grid, and less so for long-horizon planning problems, such as UnderwaterNavigation.

– There are substantial interactions between the choice of belief collection method, and the choice of value updating schedule. For example, PBVICollect and PEMACollect usually perform best with a FullUpdate, whereas HSVICollect and FSVICollect benefit from a NewestPoints update. Finally, RandomCollect tends to perform well with a PerseusUpdate. This is, for the most part, consistent with the full algorithms as presented in their respective initial publications. These observations suggest that the choice of collection method should first be matched to the type of problem at hand; the choice of backup method can then be done to match the selected belief collection method, according to the findings above.
– Most methods are robust to the size of the collected belief point set N. It is clear that N needs to be at least as large as the number of execution steps required to reach most beliefs encountered under an optimal policy. Otherwise, the search might always terminate before encountering many useful beliefs. In our experiments, we picked $N = 100$ without careful tuning; this choice was not motivated by any pre-testing or cross-validation, it simply seemed like a reasonable initial guess. Our experiments show that all methods are robust to this parameter, with the exception of PEMACollect which requires substantially fewer beliefs per rounds.
– The other key parameter in controlling the computational trade-off between adding more points and updating those that are already acquired is the number of updates, U. Our experiments show that this can usually be safely set to 1. There can be a decrease in performance when setting it to something higher (such as $U = 5$). The only exception occurred when using PEMACollect in goal-directed domains, such as RockSample and UnderwaterNavigation.
– As observed in the literature, it is preferable to use a reverse ordering of the beliefs when performing the value updates. This insight has been exploited previously in the reinforcement learning literature. More sophisticated approaches, such as eligibility traces, have not been fully investigated in the context of the point-based POMDP literature.
– We briefly experimented with removing duplicate beliefs from the set of points. This generally proves helpful, though there are a few exceptions. In general, a more systematic investigation of the role of pruning in point-based value algorithms would be highly valuable.
– Finally, we observe that it is typically worthwhile to initialize the value function with the blind policy, rather than a more naive lower bound. The computation is usually minimal (compared to the costs of the full algorithm), and therefore this can be used in general. The more important point though is that the performance boost of using a blind policy is mostly observed in domains where the blind policy is a good surrogate for the optimal policy; this suggests more generally that it is useful to initialize the value function as close as possible to the optimal policy. This is not a

new result, the literature on reward shaping has yielded similar observations (Ng et al, 1999). But here we provide some clear empirical evidence and explanations of how this impacts point-based POMDP solvers.

7.2 Limitations of analysis

While the analysis presented above examines many aspects of point-based algorithms, some of the aspects, especially pruning, are under explored. These have proven useful in some recent approaches, such as SARSOP. We did not tackle this here, as the topic is rich and deep. A systematic investigation of this topic would be highly valuable.

An important factor in scaling point-based approaches to very large domains is the use of factored, or symbolic, representations (Poupart, 2005; Shani et al, 2008b; Sim et al, 2008). This aspect is somewhat orthogonal to the questions explored here, and we did not include it to simplify the presentation and analysis. Most of the algorithmic approaches presented here can be extended to handle factored representations, through the use of appropriate data structures. We expect most of our findings to apply to this case.

Another consideration is the fact that the analysis presented above only considered components of algorithms, rather than whole algorithms. In many cases, the comparison of whole algorithms is provided in the original papers, though not often using standardized hardware and software practices. We opted for the component-based comparison because it provides new insight on the interaction between various algorithmic aspects. Nonetheless there are some disadvantages to proceeding this way, as some components that were not included may have an important effect. For example, the original HSVI algorithm updates both an upper bound and a lower bound on the value function while collecting belief points, which we did not consider here.

Finally, it is worth noting that there are other, non point-based, approaches to computing behaviors for POMDP models; some of which have achieved strong empirical results (Ji et al, 2007; Ross and Chaib-draa, 2007). Further empirical evaluation could encompass those approaches. We believe that the work presented here sets a standard for good empirical analysis in this larger context.

7.3 Point-based methods as heuristic search in the belief-MDP

It is sometimes argued that point-based method are essentially a class of MDP heuristic search methods, given that they heuristically select MDP states (POMDP belief states) and compute a value function for these states. We believe that this view is somewhat limiting — while there are many heuristic algorithms for solving MDPs, the unique structure of the belief space makes most of these algorithms either inapplicable or inefficient for the belief space MDP. For example, Shani et al (2008a) demonstrated in their prioritized value iteration algorithm how the computation of priority update, that is simple in MDPs, is difficult in POMDPs. We believe that, generally speaking, transferring ideas from MDPs to POMDPs is challenging, and may require significant changes to the original idea to apply.

As such, while it is important for the point-based research community to be knowledgeable in the latest advancements in MDP heuristic search, and consider whether techniques from MDPs can be brought into the point-based framework, we believe that many ideas that work well in general MDPs will not be easy to adapt to the belief-space MDP that interests us.

7.4 Convergence of point-based methods

Exact value iteration provably converges to the optimal value function V^* (Sondik, 1971). Some point-based algorithms, such as PBVI, HSVI, SARSOP, and GapMin, also provably converge to the optimal value function, or at least arbitrarily closely to the optimal value function, but in a slightly different sense. While exact value iteration computes the optimal value function for every possible belief state in the belief simplex, point-based algorithm typically converge to the optimal value function only over the belief space reachable from the given initial belief b_0, and in some cases the convergence of the value function is limited only to b_0. Still, an exponential number of iterations is required even for convergence on b_0. Algorithms such as HSVI, SARSOP, and GapMin, that maintain both an upper and a lower bound over the value function, can use these bounds to decide when the value function has converged and stop the value iteration.

That being said, it is widely agreed that in all the benchmarks reported in POMDP literature, point-based value iteration computes a good solution much faster than exact value iteration — typically in a few dozens of iterations. It is also obvious from the experiments that we report above that most methods find good solutions. The ADR curves also hint of a convergence to some value that could not be further improved. Indeed, additional experiments which were not described above, where some algorithms were allowed to run for much longer than the graphs above show, yielded no further improvements on these benchmarks. Still, especially in point-based algorithms that heuristically grow their belief set, it is difficult to know whether one should not expect further improvements. It is possible that the algorithm will suddenly discover a new reachable belief point where a substantial reward can be earned, thus radically changing the value function. Algorithms that maintain both an upper and a lower bound over the value function can stop once the bounds converge to within a given ϵ of each other, but experiments show that for larger domains, the gap between bounds closes very slowly, and remains considerable even after the lower bounds seems to converge (Poupart et al, 2011). Hence, the question of when is it reasonable to assume that the lower bound value function has converged in practice remains open.

Perhaps a practical way of approaching the question of when we should stop an algorithm is the anytime approach (Zilberstein, 1996), where algorithms can be stopped at any time, always returning some solution. The quality of the solution is expected to improve as more computation time and resources are given to the algorithm. Indeed, most, if not all, point-based algorithms display a good anytime behavior. As such, when to stop the algorithm is no longer a question of its convergence, but rather a function of the available time and resources. Indeed, this is the approach that was implicitly assumed in our experiments, by reporting the value function convergence over a range of time intervals.

8 Conclusion

POMDP planning algorithms have made important gains in terms of efficiency and scalability over the decade. Still, using such methods in real-world domains can be challenging, and typically requires careful design choices. The primary goal of this paper is to shed light on the various dimensions of point-based value iteration approaches.

In addition to this survey, we are publicly releasing the software used in producing the analysis presented above, containing an implementation of many point-based solvers over a uniform framework. This paper, together with the package, makes the following primary contributions:

First, we expect the survey to facilitate the targeted deployment of point-based POMDP methods in a wider variety of practical problems. In many cases the implemented algorithms can be used off the shelf to solve interesting domains specified using standard protocols. The framework also has abilities to create and solve new domains.

Second, the analysis presented here highlights opportunities for future research and development into POMDP solving algorithms. For example, we encourage investigation into methods that automatically tune their belief-space search strategy to the characteristics of the domain at hand.

Finally, by presenting a carefully designed analysis, as well as providing the related software package, we aim to encourage good experimental practices in the field. The software can be used as a platform for future algorithmic development, as well as for benchmarking different algorithms for a given problem. This can be useful to a wide range of researchers and developers in many fields.

Acknowledgments

The authors thank the anonymous reviewers for multiple helpful suggestions, both about the presentation and the experimental validation. Funding for this work was provided by the Natural Sciences and Engineering Research Council of Canada.

References

Albore A, Palacios H, Geffner H (2009) A translation-based approach to contingent planning. In: International Joint Conference on Artificial Intelligence (IJCAI), pp 1623–1628

Armstrong-Crews N, Gordon G, Veloso M (2008) Solving pomdps from both sides: Growing dual parsimonious bounds. In: AAAI workshop for Advancement in POMDP Solvers

Atrash A, Kaplow R, Villemure J, West R, Yamani H, Pineau J (2009) Development and validation of a robust speech interface for improved human-robot interaction. International Journal of Social Robotics 1:345–356

Barto AG, Bradtke SJ, Singh SP (1995) Learning to act using real-time dynamic programming. Artificial Intelligence 72:81–138, DOI 10.1016/0004-3702(94)00011-O

Bellman R (1957a) Dynamic programming. In: Princeton University Press

Bellman R (1957b) A Markovian decision process. Journal of Mathematics and Mechanics 6:679–684

Bonet B, Geffner H (2003) Labeled RTDP: Improving the convergence of real-time dynamic programming. In: International Conference on Planning and Scheduling (ICAPS), pp 12–31

Bonet B, Geffner H (2009) Solving POMDPs: RTDP-Bel vs. Point-based algorithms. In: International Joint Conference on Artificial Intelligence (IJCAI), pp 1641–1646

Boutilier C (2002) A POMDP formulation of preference elicitation problems. In: National Conference on Artifical Intelligence (AAAI), pp 239–246

Brunskill E, Kaelbling L, Lozano-Perez T, Roy N (2008) Continuous-state pomdps with hybrid dynamics. In: International Symposium on Artificial Intelligence and Mathematics (ISAIM)

Cassandra A, Littman ML, Zhang NL (1997) Incremental Pruning: A simple, fast, exact method for partially observable Markov decision processes. In: Conference on Uncertainty in Artificial Intelligence (UAI), pp 54–61, URL `http://www.cs.duke.edu/\~mlittman/docs/uai97-pomdp.ps`

Dai P, Goldsmith J (2007) Topological value iteration algorithm for Markov decision processes. In: International Joint Conference on Artificial Intelligence (IJCAI), pp 1860–1865

Dibangoye JS, Shani G, Chaib-draa B, Mouaddib AI (2009) Topological order planner for POMDPs. In: International Joint Conference on Artificial Intelligence (IJCAI), pp 1684–1689

Doshi F, Roy N (2008) The permutable POMDP: fast solutions to POMDPs for preference elicitation. In: International Conference on autonomous Agents and Multiagent Systems (AAMAS), pp 493–500

Geffner H, Bonet B (1998) Solving large POMDPs using real time dynamic programming. In: Proceedings AAAI Fall Symp. on POMDPs

Hansen E (1998) Solving POMDPs by searching in policy space. In: Conference on Uncertainty in Artificial Intelligence (UAI), pp 211–219

Hansen EA (2007) Indefinite-horizon POMDPs with action-based termination. In: National Conference on Artificial Intelligence (AAAI), pp 1237–1242

Hauskrecht M (1997) Incremental methods for computing bounds in partially observable Markov decision processes. In: National Conference on Artificial Intelligence, pp 734–739

Hauskrecht M (2000) Value-function approximations for partially observable Markov decision processes. Journal of Artificial Intelligence Research (JAIR) 13:33–94, URL `http://www.cs.washington.edu/research/jair/abstracts/hauskrecht00a.html`

Hauskrecht M, Fraser HSF (2000) Planning treatment of ischemic heart disease with partially observable markov decision processes. Artificial Intelligence in Medicine 18(3):221–244

Hoey J, Poupart P, von Bertoldi A, Craig T, Boutilier C, Mihailidis A (2010) Automated handwashing assistance for persons with dementia using video and a partially observable markov decision process. Computer Vision and Image Understanding 114(5):503–519

Howard RA (1960) Dynamic Programming and Markov Processes. MIT Press, Cambridge, Massachusetts

Hsiao K, Kaelbling LP, Lozano-Pérez T (2007) Grasping POMDPs. In: IEEE International Conference on Robotics and Automation (ICRA), pp 4685–4692

Huynh VA, Roy N (2009) icLQG: Combining local and global optimization for control in information space. In: IEEE International Conference on Robotics and Automation (ICRA), pp 2851–2858

Izadi MT, Rajwade AV, Precup D (2005) Using core beliefs for point-based value iteration. In: International Joint Conference on Artificial Intelligence, pp 1751–1753

Izadi MT, Precup D, Azar D (2006) Belief selection in point-based planning algorithms for POMDPs. In: Canadian Conference on Artificial Intelligence, pp 383–394

Ji S, Parr R, Li H, Liao X, Carin L (2007) Point-based policy iteration. In: National conference on Artificial intelligence (AAAI), AAAI Press, pp 1243–1249

Kaelbling L, Littman M, Cassandra A (1998) Planning and acting in partially observable stochastic domains. In: Artificial Intelligence, pp 99–134

Kaplow R (2010) Point-based pomdp solvers: Survey and comparative analysis. Master's thesis, McGill University

Kurniawati H, Hsu D, Lee W (2008) SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In: Robotics: Science and Systems (RSS)

Littman ML (1996) Algorithms for sequential decision making. PhD thesis, Department of Computer Science, Brown University, Providence, RI, URL `ftp://ftp.cs.brown.edu/pub/techreports/96/cs96-09.ps.Z`, also Technical Report CS-96-09

Littman ML, Cassandra AR, Kaelbling LP (1995) Learning policies for partially observable environments: Scaling up. In: International Conference on

Machine Learning (ICML), pp 362–370

Littman ML, Sutton RS, Singh SP (2001) Predictive representations of state. In: Advances in Neural Information Processing Systems (NIPS), pp 1555–1561

Littman ML, Ravi N, Fenson E, Howard R (2004) An instance-based state representation for network repair. In: National Conference on Artifical Intelligence (AAAI), pp 287–292

Lovejoy WS (1991) Computationally feasible bounds for partially observed Markov decision processes. Operations Research 39(1):162–175

Ng A, Harada D, Russell S (1999) Policy invariance underreward transformations: Theory and application to reward shaping. In: International Conference on Machine Learning (ICML)

Pineau J, Gordon G (2005) POMDP planning for robust robot control. In: International Symposium on Robotics Research (ISRR), Springer, vol 28, pp 69–82

Pineau J, Gordon G, Thrun S (2003a) Point-based value iteration: An anytime algorithm for POMDPs. In: International Joint Conference on Artificial Intelligence, pp 1025–1032

Pineau J, Gordon GJ, Thrun S (2003b) Applying metric-trees to belief-point POMDPs. In: Advances in Neural Information Processing Systems (NIPS)

Pineau J, Gordon GJ, Thrun S (2006) Anytime point-based approximations for large pomdps. Journal of Artificial Intelligence Research (JAIR) 27:335–380

Poon L (2001) A fast heuristic algorithm for decision theoretic planning. Master's thesis, The Hong-Kong University of Science and Technology

Porta JM, Vlassis N, Spaan MTJ, Poupart P (2006) Point-based value iteration for continuous POMDPs. Journal of Machine Learning Research 7:2329–2367

Poupart P (2005) Exploiting structure to efficiently solve large scale partially observable markov decision processes. PhD thesis, Department of Computer Science, University of Toronto

Poupart P, Boutilier C (2003) Bounded finite state controllers. In: Advances in Neural Information Processing Systems (NIPS)

Poupart P, Kim KE, Kim D (2011) Closing the gap: Improved bounds on optimal POMDP solutions. In: International Conference on Planning and Scheduling (ICAPS)

Puterman ML (1994) Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc., New York, NY, USA

Ross S, Chaib-draa B (2007) AEMS: An anytime online search algorithm for approximate policy refinement in large POMDPs. In: International Joint Conference on Artificial Intelligence (IJCAI), pp 2592–2598

Sanner S, Kersting K (2010) Symbolic dynamic programming for first-order POMDPs. In: National Conference on Artificial Intelligence (AAAI)

Shani G (2010) Evaluating point-based POMDP solvers on multicore machines. IEEE Transactions on Systems, Man, and Cybernetics, Part B 40(4):1062–1074

Shani G, Meek C (2009) Improving existing fault recovery policies. In: Advances in Neural Information Processing Systems (NIPS), vol 22, pp 1642–1650

Shani G, Heckerman D, Brafman RI (2005) An MDP-based recommender system. Journal of Machine Learning Research 6:1265–1295

Shani G, Brafman R, Shimony S (2007) Forward search value iteration for POMDPs. In: International Joint Conference on Artificial Intelligence (IJCAI)

Shani G, Brafman RI, Shimony SE (2008a) Prioritizing point-based POMDP solvers. IEEE Transactions on Systems, Man, and Cybernetics, Part B 38(6):1592–1605

Shani G, Poupart P, Brafman RI, Shimony SE (2008b) Efficient ADD operations for point-based algorithms. In: International Conference on Automated Scheduling and Planning (ICAPS), pp 330–337

Sim HS, Kim KE, Kim JH, Chang DS, Koo MW (2008) Symbolic heuristic search value iteration for factored POMDPs. In: National Conference on Artificial intelligence, pp 1088–1093

Singh SP, Sutton RS (1996) Reinforcement learning with replacing eligibility traces. Machine Learning 22:123–158

Smith T, Simmons R (2004) Heuristic search value iteration for POMDPs. In: Conference on Uncertainty in Artificial Intelligence (UAI)

Smith T, Simmons RG (2005) Point-based POMDP algorithms: Improved analysis and implementation. In: Conference on Uncertainty in Artificial Intelligence (UAI), pp 542–547

Sondik E (1971) The optimal control of partially observable Markov decision processes. PhD thesis, Stanford University

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

Spaan M, Vlassis N (2004) A point-based POMDP algorithm for robot planning. In: IEEE International Conference on Robotics and Automation (ICRA), pp 2399–2404

Spaan M, Vlassis N (2005) Perseus: Randomized point-based value iteration for POMDPs. In: Journal of Artificial Intelligence Research, pp 195–220

Sutton RS, Barto AG (1998) Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA

Szepesvari C (2009) Reinforcement learning algorithms for MDPs - a survey. Tech. Rep. TR09-13, University Of Alberta

Virin Y, Shani G, Shimony SE, Brafman RI (2007) Scaling up: Solving POMDPs through value based clustering. In: National Conference on Artificial Intelligence (AAAI), pp 1290–1295

Wang C, Khardon R (2010) Relational partially observable mdps. In: National Conference on Articial Intelligence (AAAI)

Williams JD, Young S (2007) Partially observable markov decision processes for spoken dialog systems. Computer Speech & Language 21(2):393–422

Wingate D, Seppi KD (2005) Prioritization methods for accelerating mdp solvers. Journal of Machine Learning Research (JMLR) 6:851–881

Zhang NL, Zhang S (2001) Speeding up the convergence of value iteration in partially observable Markov decision processes. Journal of Artificial Intelligence Research (JAIR) 14:29–51

Zilberstein S (1996) Using anytime algorithms in intelligent systems. AI Magazine 17:73–83