Value Iteration Algorithms for POMDPs

Raihan Seraj and Samin Yeasar Arnob

August 19, 2019

Abstract: In this work we make an analysis of value iteration algorithms for Partially Observable Markov Decision Process (POMDPs). We mainly discuss the structure of value function for POMDPs and the difficulties associated with performing value iteration for POMDPs with large state spaces. We further discuss two algorithms for performing value iteration namely, the Witness Algorithm and Point Based Value Iteration Algorithm and perform a comparative analysis of both of these methods.

1 Introduction

Markov decision processes [1] is a mathematical framework, where the objective of a decision maker, (i,e an *agent*) is to act in order to maximize expected *rewards* in a defined *environment*. An important aspect about MDP model is that, given a stochastic model of the environment and a goal, it provides a basis for algorithms that seeks to find an optimal policy. Where a policy π is a mapping from states to actions. In an MDP framework it's assumed that the agent has complete information about it's current state. In contrast when an agent acts in real environments, the assumption of environment being completely deterministic often fails even in low-level control problems. A partially observable Markov decision process (POMDP) is a generalization of a Markov decision process (MDP). In POMDP models an agent can not directly observe the underlying state. The POMDP framework is a model for planning under uncertainty [2] [3]. However,solving for a POMDP model becomes intractable even for small problems due to their complexity.

In the following section, we give an overview of the underlying dynamics of a POMDP model and highlight its main differences from an MDP formulation. We discuss the idea and talk about relevant literatures that discuss the structure of finite horizon value function of POMDPs. We outline the difficulties of performing exact value iteration updates, and discuss the notion of α vectors associated with the value function for POMDPs. We then talk about two value iteration algorithms (the Witness Algorithm [4] [5] and Point Based Value Iteration Algorithm [6] in detail and illustrate how these algorithms solve the complexity of value iteration for POMDPs to make computation more efficient.

2 An overview of POMDP

In this section we begin our explanation of basic POMDP formalism. POMDPs handle uncertainty in both action effects and state observability, whereas many other frameworks handle neither of these, and some handle only stochastic action effects. To handle partial state observability, plans are expressed over information states, instead of world states, since the latter ones are not directly observable. The space of information states is the space of all beliefs a system might have regarding the world state.

2.1 POMDP Model

A POMDP can be represented using the following n-tuple: $\{S, A, Z, b_o, T, \Omega, R, \lambda\}$, where

- S is a (finite) set of discrete states, that captures all information relevant to agents decision making process.
- A is a set of discrete actions, which stochastically affect the state of the world. Choosing the right action as a function of history is the core problem in POMDPs
- $T : S \times A \times S' \to [0,1]$ is the transition function, where $T(s,a,s') := Pr(s_{t+1} = s | a_t = a, s_t = s)$ represents the distribution describing the probability of transitioning from state s to state s'.
- $R : B \times A$ is the reward function received when executing action, a in belief state, b.
- Z is a set of discrete observations providing incomplete and/or noisy state information.
- $\Omega : S \times A \times Z \to [0,1]$is the observation function, where $\Omega(z,s,a) := Pr(z_{t+1} = z | a_t = a, s_{t+1} = s)$, the distribution describing the probability of observing z from state s after taking action a.

In POMDP model, states are not directly observable, rather the agent has access to observations which gives noisy information about the state. The belief state is a probability distribution over all possible states $s \in S$, and the entire probability space (the set of all possible probability distributions) in known as the belief space Π.

An information state $\mathbb{M}_t$ is a function of the information, I (more precisely is measurable with respect to the observation sigma algebra). The information state is denoted as $\mathbb{M}_t = \mathscr{F}(I)$. An information state therefore satisfies the following properties

- Is sufficient for predicting itself
  $\mathbb{P}(\mathbb{M}_{t+1} = m_{t+1} | I_t = i_t, A_t = a_t) = \mathbb{P}(\mathbb{M}_{t+1} = m_{t+1} | \mathbb{M}_t = \mathscr{F}(I_t), A_t = a_t)$
- Is sufficient for performance evaluation
  $\mathbb{E}[C_t | I_t = i_t, A_t = a_t] = \mathbb{E}[C_t | \mathbb{M}_t = \mathscr{F}_t(i_t), A_t = a_t]$

A belief state follows both the conditions hence it is an information state. The belief state formalism provides a sufficient statistic for an agent to act optimally [11].

A belief state, $b \in \Pi(s)$, represents the agent's current state given its all previous action and observations.

$$b_t(s) = P(S_t = s | Z_{1:t} = z_{1:t}, A_{1:t-1} = a_{t-1})$$

For any state $s \in S$, belief state $b[s] \in [0,1]$ and $\sum_{s \in S} b[s] = 1$. At each time step belief state is updated for each action and observation using Bayes rule as shown by the following equations.

$$\begin{aligned}
b'[s] &= P(s' | z, a, b) \\
&= \frac{P(z, s' | a, b)}{P(z | a, b)} \\
&= \frac{P(z | s', a, b) * P(s' | a, b)}{P(z | a, b)} \\
&= \frac{p(z | a, s') \sum_{s' \in S} P(s' | a, s) * b(s)}{P(z | a, b)} \\
&= \frac{\Omega(z, a, s') \sum_{s' \in S} T(s', a, s) * b(s)}{P(z | a, b)}
\end{aligned} \tag{1}$$

2.2 Value function for POMDP

The objective of POMDP planning is to find sequential actions $\{a_0, a_1 \dots a_t\}$ maximizing expected sum of discounted rewards $\mathbb{E}[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)]$, where γ is the discount factor. As the true states of the environment are not directly observable by the agent in POMDP, the objective then becomes maximizing the expected discounted rewards over the probability distribution over each of the states which is known as the belief state. Hence the value function at a particular belief state can be represented as:

$$V_t(b) = \max_{a \in A} \Big[R(b,a) + \gamma \mathbb{E}\big[V_{t+1}(b')\big] \Big] \tag{2}$$

Now,

$$R(b,a) = \sum_{s \in S} b[s] R[s,a] \tag{3}$$

Where, R[s,a] is the immediate reward received by an agent in state s for taking action a and b[s] is the probability of the agent being at state s. $\sum_{s \in S} b[s] R[s,a]$ is the probability weighted sum of immediate reward.Thus value function can be represented as

$$V_t(b) = \max_{a \in A} \Big[\sum_{s \in S} b[s] R[s,a] + \gamma \sum_{z \in Z} P(z|b,a) V_{t+1}(b') \big] \Big] \tag{4}$$

$\sum_{z \in Z} P(z|b,a) V(b')$ implies the probability weighted average value of the resulting belief state.

3 POMDP and requirement of α vector

In POMDP we get a continuous value function over belief space. For finite number of states,we get infinite number of possible belief states in POMDP. Solving for single belief becomes intractable. Hence, due to infinite number of value functions at each time-step, it is impossible to do traditional DP.

To understand the problem we can consider an example in figure 1. Where we have two states. S0 and S1 . For two state case belief space is a single line and probability of being at any of the states is maximum at the terminals. At terminal point 0 probability of being at state S0 is 1 and probability of being at state S1 is 0. Any point between [0,1] represents a probability distribution over state space. Even though we have only two states, number of belief states are infinite. Here, value function over belief space in continuous. This continuity makes solving POMDPs harder than solving MDPs.

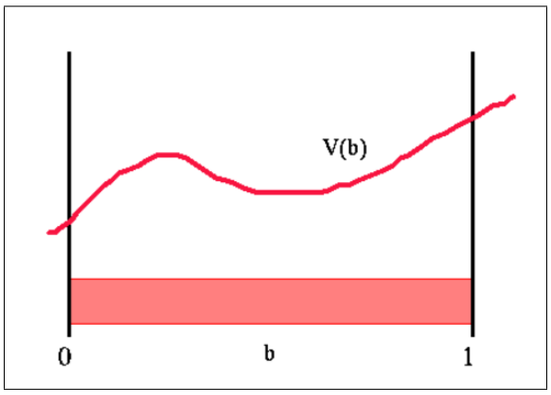

Figure 1: Belief simplex for 2 states. source: www.pomdp.org

A fundamental fact about finite horizon value function is that it is piecewise linear and convex when we are maximizing reward (or concave when we are minimizing cost) for every horizon length [10]. Which basically confirms, we can represent continuous function with set of linear functions where the linear functions can handle infinite number of belief inputs. By doing this we are representing infinite number of belief states in finite number of linear functions.

For two states example in figure 2, the value function was possible to represent in terms of combination of lines. But when we have more than two states the value functions are represented as combination of hyperplanes. Thus value function V_t at any horizon t is represented as set of $|S|$ dimensional vectors known as alpha vector : $\Gamma = \{\alpha_1, \alpha_2 \dots \alpha_m\}$. Number of elements in each vector α_i equivalent to number of states. Where α_i vectors represent the co-efficients of the hyperplanes.

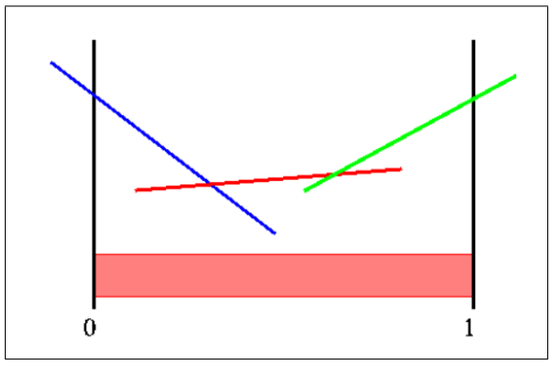

Figure 2: Discretized belief space source: www.pomdp.org

The value at any particular belief is calculated by evaluating the belief over each of the linear function. This is simply done by taking the dot product of each of the α vectors with the belief state and maximizing over them.

As we represent, the hyper-plane as a vector $\{\alpha_1, \alpha_2 \dots \alpha_m\}$ (i.e., the equation coefficients) and each belief state as a vector $(b_1, b_2, b_3, \dots)$ (the probability at each state) then optimal t-horizon value function can be written as:

$$V_t(b) = \max_{\alpha \in \Gamma_t} \sum_{s \in S} b(s)\alpha(s) \tag{5}$$

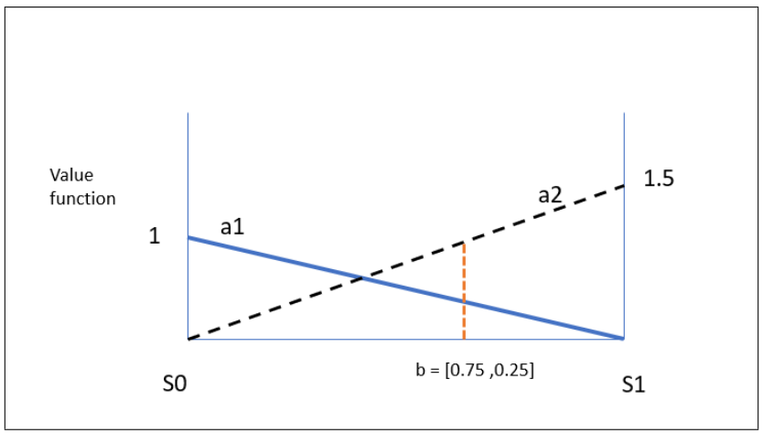

Figure 3: Discretized belief space source

We can consider an example (figure 3) to get a clear view of the value function computation. In figure 3, we are representing our value function with combination of two linear functions a_1, a_2. At any belief point over the belief space we have two possible options to represent our value function. For a given belief [0.75, 0.25], our value function corresponding to our actions would be:

$$\begin{aligned} V^{a_1}(b) &= [0.75, 0.25] \cdot [0, 1] \\ &= 0.25 \\ V^{a_2}(b) &= [0.75, 0.25] \cdot [1.5, 0] \\ &= 1.125 \\ \therefore V(b) &= max[V^{a_1}(b), V^{a_1}(b)] \\ &= 1.125 \end{aligned}$$

Where $[0, 1]$ are coefficients $([\alpha_1^1, \alpha_2^1])$ of linear function a_1 and $[1.5, 0]$ are coefficients $([\alpha_1^2, \alpha_2^2])$ of linear function a_2.

We can write the value of the belief update b' as:

$$\begin{aligned} V(b') &= \max_{\alpha \in \Gamma} \sum_{s \in S} \alpha(s) b'(s) \\ &= \max_{\alpha \in \Gamma} \sum_{s \in S} \alpha(s) \frac{\Omega(z|a,s') \sum_{s' \in S} T(s', a, s) * b(s)}{P(z|a,b)} \end{aligned} \tag{6}$$

We can unroll the value function in the following manner:

$$V_t(b) = \max_{a \in A} V_t^a(b) \tag{7}$$

$$V_t^a(b) = \sum_{z \in Z} V_t^{a,z}(b) \tag{8}$$

$$V_t^{a,z}(b) = \sum_{s \in S}(R(s,a) * b(s) + \gamma \max_{\alpha \in \Gamma} \sum_{s \in S} \sum_{s' \in S} P(z|a,b)\alpha(s)\frac{\Omega(z,a,s')T(s,a,s') * b(s)}{P(z|a,b)} \tag{9}$$

$$= \sum_{s \in S}(R(s,a) * b(s) + \gamma \max_{\alpha \in \Gamma} \sum_{s \in S} \sum_{s' \in S} \alpha(s)\Omega(z,a,s')T(s,a,s') * b(s) \tag{10}$$

The t^{th} horizon value function can be updated from the V_{t+1} using *backup operator*, H. Where exact update $V = HV'$. Many algorithms [12] [13] directly manipulates α vectors using combination of set projection and pruning. We can define straight-forward exact POMDP value update. **(8)** can be represented as cross sum of bag of vectors for each observation *(step 1)*:

$$\Gamma_t^a = \bigoplus_z \Gamma_t^{a,z} \tag{11}$$

(7) can be represented as union over bag of vectors for each of the action *(step 2)*:

$$\Gamma_t = \cup_a \Gamma_t^a \tag{12}$$

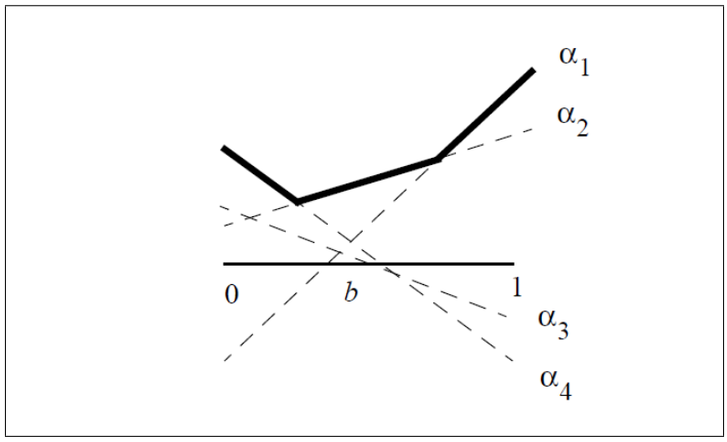

Figure 4: For given action,a we get bag of vector $\{\alpha_1, \alpha_2, \alpha 3, \alpha 4\}$. Source:[5]

Even though we get finite number of α vectors, with each value iteration new α vectors get added at exponential rate. Thus value iteration gets really expensive. In, practice, many of the vectors in final set Γ may be completely dominated by another vector ($\alpha_i.b < \alpha_j.b, \forall b \in B$), or by combination of other vectors. For example in figure 4, we can see that we can create the convex just using first three *alpha* vectors and α_4 is not needed. So, We can discard α_4 without affecting the solution and this is called *pruning*. To understand the need for pruning, let's $|V|$ be the number of α vector in previous solution set. Step 1 generates $|A||V|^{|Z|}$ cross sum. For the worst case scenario updated value function will be the size of $|A||V|^{|Z|}$(times$|S|^2|A||V|^{|Z|}$). Even though it may need additional computations to find the dominant vectors, the exponential growth of α vectors makes it necessary to perform the pruning operation. Another option could be, representing a model with simplest possible combinations of α vectors and see if it completely represents value function for all the beliefs. If not, then new α vectors can be added.

In the next section we discuss about the Witness algorithm that performs some sophisticated mechanism in order to find optimal solution of value iteration.

4 The Witness Algorithm

The Witness algorithm proposed in [4] uses an approach of finding the exact solution to value iteration algorithms in POMDPs. The main idea of the algorithm is to construct an α vector for each action and observation given a belief point. It then considers the region over which this vector is dominant, and looks for evidence (i. e. a Witness point) where the vector is suboptimal. When it finds such a point, it can generate the best vector at that point until no new witnesses are found. This produces an optimal solution.

4.1 Policy Tree

A t-step policy can be completely described by a choice of action and one $t-1$ step policy for each observation. If p is a t-step policy tree, action(p)$\in A$ defines initial action choice and choice(p,z) would be $t-1$ step policy tree followed if z is observed after the first step. Figure 5 illustrates the definition.

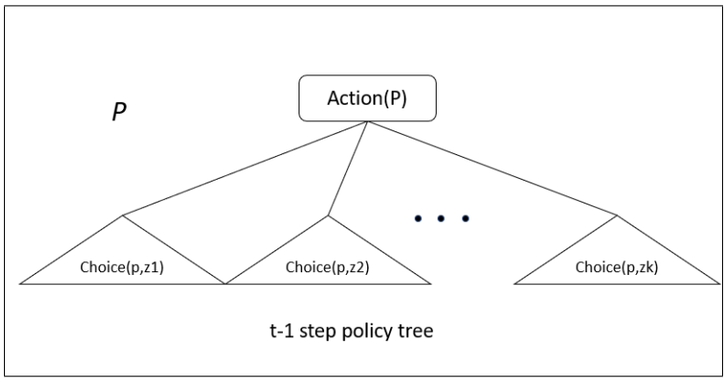

Figure 5: Notation used for describing policy tree

Instead of computing V_t directly, in witness algorithm, tries to build a collection of policy trees that represents Q_t^a for each $a \in A$. $Q_t^a(b)$ is the expected reward for taking action a from and observing z from belief state b. As value function is piece-wise linear and convex and can be represented by collection of policy tree, Q_t^a which corresponds to action value function can also be represented by collection of policy trees.

The policy trees in $\hat{Q}_t^a$ divide belief space into regions. A policy tree is just the set of belief states over which it is optimal. Initially in witness algorithm, simplest collection of policy trees are used to represent value function. For any belief state b, if there exist a better policy tree for an observation, then b is witness. From simple combination, more policy trees are added. In another word the belief space is divided into more regions. And if none individual choice is improved, then there are no witness. Continuous belief space is partitioned into finite number of regions.

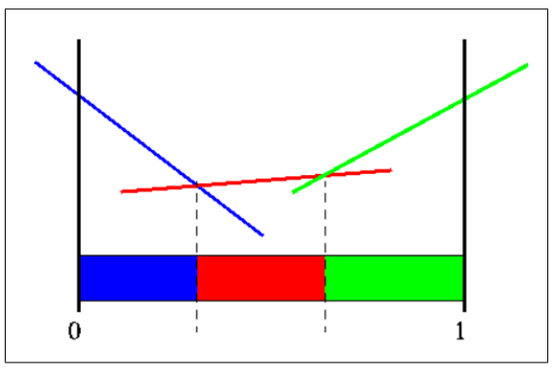

Figure 6: Sample PCWL function and it's partition in belief space. Source: http://www.pomdp.org/

4.2 Basic Approach

Witness algorithm tries to find minimal set of policy tree for representing Q_t^a for each action a . Considering the iteration for each Q_t^a-functions one at a time.

A minimum set of representative policy tree for Q_t^a is found for each of the actions. The set Γ_a of policy trees is initialized with a single policy tree with an action a being the root of the tree. This action is the best action at some arbitrary belief state. Given that the set of policy trees $\mathscr{V}_t$ to represent the value function is a subset of the set of policy trees Γ_t^a that represents the action value function at time t. This is represented by the following

$$\mathscr{V}_t \subseteq \bigcup_a \Gamma_t^a$$

At each iteration using one step lookahead, the algorithm computes the true $Q_t^a(b)$ which is obtained from the vector set $\mathscr{V}_{t-1}$ that represents the value function at $(t-1)^{th}$ step. The algorithm then checks for the existence of some beliefs for which $\hat{Q}_t^a(b)$, the approximated action value function computed from the set of policy trees Γ^a is equal to the exact $Q_t^a(b)$ computed from one step lookahead function. The existence of such beliefs is regarded as a witness point because it testifies about the fact that the current set of policy trees $\Gamma^a$is not an exact representative of $\hat{Q}_t^a(b)$

Once a witness point is identified a policy tree should be obtained with an action a at its root that results in the best value at that belief state. To construct such a policy tree at time t, the algorithm needs to incorporate the $(t-1)$ policy trees that corresponds to each observation o and execute these policy trees if an observation is made after taking the action a. This means that when the agent will be in b'(i,e next belief calculated from the Bayesian update rule) from which it should execute (t-1) step policy tree $p_o \in \mathscr{V}_{t-1}$ that maximizes $V_{p_o}(b)$ Here p_o is just a subtree that is built for each observation o. The new policy tree is thus added to Γ^a uptill no more witness point exists thereby ensuring that the $\hat{Q}_t^a(b) = Q_t^a(b)$[14]

The Witness Algorithm therefore finds exact solutions to discounted finite horizon POMDPs using value iteration. After its k-th iteration the algorithm returns the exact k-step Q functions as collections of vectors. Γ_a for each actions. The algorithm can be used to find arbitrarily accurate approximations to optimal infinite horizon Q functions.

We therefore outline the linear program that witness algorithm solves each time to find an evidence whether the approximated value function is equal to the exact. Algorithm therefore solves the following linear program.[14]

$$\begin{aligned}
&\text{Inputs:}\\
&\Gamma_a, p_{new}\\
&\text{Variables:}\\
&\delta, b(s) \text{for each } s \in S\\
&\text{Maximize} \quad \delta\\
&\text{Improvement Constraints:}\\
&\text{For each } \tilde{P} \text{ in } \Gamma_a : V_{p_{new}}(b) - V_{\tilde{p}}(b) \geq \delta\\
&\text{Simplex Constraints:}\\
&\text{For each } s \in S : b(s) \geq 0\\
&\sum_{s \in S} b(s) = 1
\end{aligned}$$

In the linear Program discussed above, the inputs to the program includes the current policy tree for estimating $Q_t^a(b)$ the variables include the δ $b(s)$ and the states $s \in S$. The variable δ is the minimum amount of improvement of p_{new} over any policy tree in Γ^a at b The algorithm then seeks to maximize the advantage of p_{new} over all $\tilde{p} \in \Gamma^a$.

Experiments performed in [4] and [5] illustrates that the witness algorithm is faster in practice over a wide range of problem size. the algorithm runs in polynomial time in the number of policy trees compared to other algorithms which (if they construct policy trees) will have an exponential running time.

5 Point Based Value Iteration

The Point Based Value Iteration (PBVI) was introduced by [6] in order to perform approximate value iteration algorithm for POMDPs with large state spaces.PBVI specifically targets the curse of history and shares many similarities with earlier grid based methods described in [7][8] The algorithm provides theoretical guarantees of finding an upper bound to the approximation. PBVI selects a small set of representative belief points and maintains α vectors for each of these belief points so as to keep the piecewise linearity and convexity of the finite horizon value function. The algorithm then performs value backup to these points and updates both the value and the value gradient which provides better generalization to unexplored belief[9].

5.1 Basic Approach

The Point Based Value Iteration, is regarded as an anytime algorithm. Which means that a trade off is done in terms of the accuracy and the computation time in this algorithm. The algorithm interleaves between steps of value iteration and steps of belief set expansion.

The point based value iteration performs a point based backup according to the following equation

Given the α vectors for $t-1$ horizon, the α vector corresponding to the t^{th} horizon is constructed as the follows for a given belief b. Thus the following sets are defined

$$\Gamma^{a,*} = R(s,a)$$
$$\Gamma^{a,z} \leftarrow \alpha_i^{a,z}(s) = \gamma \sum_{s' \in S} T(s,a,s') O(s',a,z) \alpha_i'(s'), \forall \alpha_i' \in \Gamma_{t-1}$$

Where Γ_{t-1} is the set of α vectors at time $t-1$ and $\Gamma^{a,z}$ is the set of α vectors for each action and observation Thus now instead of searching through the whole belief space and perform an exact update which requires cross sum, the following summation is performed since the set of representative belief points are now finite

$$\Gamma_b^a = \Gamma^{a,*} + \sum_{z \in Z} \arg\max_{\alpha \in \Gamma^{a,z}} \big(\sum_{s \in S} \alpha(s) b(s) \big)$$

Hence the best action for each belief point is found as

$$\Gamma_t \leftarrow \arg\max_{\Gamma_b^a, \forall a \in A} \Big(\sum_{s \in S} \Gamma_b^a(s) b(s) \quad \forall b \in B$$

Therefore after obtaining the α vectors the value function at each belief is calculated by the following equation

$$V_t(b) = \max_{\alpha \in \Gamma_t} \sum_{s \in S} (\alpha(s) b(s)$$

The exact algorithm for performing point based value backup is given as follows.

Γ_t=BACKUP(B, Γ_{t-1})	1
For each action $a \in A$	2
For each observation $z \in Z$	3
For each solution vector $\alpha' \in \Gamma_{t-1}$	4
$\alpha^{a,z}(s) = \gamma \sum_{s' \in S} T(s,a,s') O(s',a,z) \alpha'(s'), \forall s \in S$	5
$\Gamma^{a,z} \leftarrow \alpha^{a,z}$	6
End	7
End	8
End	9
For each belief point $b \in B, \forall s \in S$	10
$\alpha_b \leftarrow \text{argmax}_{a \in A} \left[\sum_{s \in S} R(s,a) b(s) + \sum_{z \in Z} \max_{\alpha \in \Gamma^{a,z}} \left[\sum_{s \in S} \alpha(s) b(s) \right] \right]$	11
If($\alpha_b \notin \Gamma_t$)	12
$\Gamma_t \leftarrow \alpha_b$	13
End	14
Return Γ_t	15

Figure 7: Point Based Value Backup

The algorithm starts with a small initial set of belief points and applies a series of backup operator as shown in the figure above. The set of belief points is grown gradually which is known as the belief set expansion step. Once the set of belief points are grown this backup operation is repeated for both the new and the old set of points all over again. Thus one can obtain an approximate value function for a fixed number of belief set expansions N

For a finite time horizon T, the backup operation is done T times in case of infinite horizon problems, the horizon is selected by the condition $\gamma[R_{max} - R_{min}] < \varepsilon$ where $R_{max} = \max_{a,s} R(s,a)$ and $R_{min} = \min_{a,s} R(s,a)$ and ε is some tolerance term.

Once a t horizon value backups are performed for particular set of representative belief points, the belief points are expanded by sampling belief points from stochastic trajectories. The reachable belief points are chosen by simulating a single step forward trajectory for the given belief $b \in B$ This means that for each action observation pair, a new set of belief points $\{b_0, b_1, ..b_n\}$ are obtained using the Bayesian update rule. However not all the points are included in the existing set of belief points. PBVI calculates the L_1 from all the points in the existing belief set to these new set of belief points and include those points whose distance are the furthers. This operation is performed for each action and the observation pair. One can use both the L1 and L2 distance measure in order to incorporate the reachable belief points in the new set.

The algorithm presented above does not require an explicit pruning step that requires to solve an LP in order to find the dominant vectors, the steps 12 and 13 in the algorithm shows that pruning is trivial in the algorithm itself.

5.2 Error Bounds of PBVI

The PBVI have theoretical guarantees on the maximum approximation error the algorithm can make in worst case. The error is upper bounded by $\frac{R_{max}-R_{min}}{1-\gamma}$ We therefore outline the proof of this error bound as described in the paper.

Considering PBVI making its worst case value update in the belief point b' and let $b \in B$ be the closest 1-norm sampled belief to b'.Let α be the vector that is maximal at b and α' be the vector that would be maximal at b' Hence by failing to include α' in the solution set PBVI makes an error of at most $\alpha' \cdot b' - \alpha \cdot b$ On the other hand since α is maximal at b then $\alpha' \cdot b \leq \alpha \cdot b$

$$
\begin{aligned}
\varepsilon &\leq \alpha' \cdot b' - \alpha \cdot b \\
&= \alpha' \cdot b' - \alpha \cdot b' + (\alpha' \cdot b - \alpha' \cdot b) \quad \text{add zero} \\
&\leq \alpha' \cdot b' - \alpha \cdot b' + \alpha \cdot b - \alpha' \cdot b \quad \alpha \text{ optimal at B} \\
&= (\alpha' - \alpha) \cdot (b' - b) \quad \text{collect terms} \\
&\leq \|\alpha' - \alpha\|_\infty \|b' - b\|_1 \quad \text{Holder inequality} \\
&\leq \|\alpha' - \alpha\|_\infty \delta_B \quad \text{definition of } \delta_B \\
&\leq \frac{(R_{max} - R_{min})\delta_B}{1-\gamma}
\end{aligned}
$$

In the above proof the last inequality holds because each α vector represents the reward achievable starting from some state and following some sequence of actions and observations. The sum of rewards therefore must fall between $\frac{R_{min}}{1-\gamma}$ and $\frac{R_{max}}{1-\gamma}$

6 Critical and Comparative Analysis

In this section we perform a comparative analysis of both the algorithms discussed above, we make detail comparison between both the algorithms and outline the shortcomings of each of these methods. Table 1 gives an overview of the differences of the two algorithms

Table 1: Comparative analysis of Witness Algorithm and PBVI

Witness Algorithm	**PBVI**
Finds a witness point that suffices whether the approximate value is close to the optimal. Thereby finds an exact value	Finds an approximation to the value function upper bounded by $\frac{(R_{\max}-R_{\min})\varepsilon_B}{(1-\gamma)^2}$
Suitable for solving small problems	Can handle problems with large state space and tackles the problem of curse of dimensionality
Needs to solve an LP each time to decide whether to include the vectors in the current set (pruning)	The algorithm itself includes a trivial pruning step where only those α vectors are included not present in the current set.
Find places where value function is suboptimal, Operates action by action and observation by observation to build up α in exponential time	Updates are done in polynomial time. Belief update is modified one vector is maintained at each point.
Construct policies that cover the entire belief space	Considers reachable belief states and performs value backups for each state
Solutions are only obtained untill no witness points are found for all set of belief states	Solutions can be obtained any time with a trade off between computation power and the quality of the solution

Comparison of runtime and complexity

The witness algorithm improves the complexity of the value iteration algorithm which generates all the α vectors and then prunes them. It does so by calculating the set of α vectors Γ^a that corresponds to action value function Q_t^a rather than the α vector set $\mathscr{V}_t$ that corresponds to the value function only. Thus the algorithm computes the vectors Γ^a in time polynomial in $|S|, |A|, \Omega, \mathscr{V}_{t-1}$ where the Ω is a set of observations and $\mathscr{V}_{t-1}$ is the set of policy trees or the alpha vectors representing the value function at $t-1$ time step.

A single point based update in PBVI creates $|A|, |\Omega|, \Gamma_{t-1}^a$ projections in time $|S|^2|A||\Omega||\Gamma_{t-1}|$ in the first step. The consecutive steps reduce the set to at most $|B|$ components where the set B contains the reachable belief points[9]. Thus a full point based value update runs in polynomial time and most importantly the size of the Γ_t remains constant at every iteration. This is in contrast to the complexity of the witness algorithm whether the size of the set Γ_t^a grows, thus to point specifically, if the set Γ_t^a grows exponentially, then the witness algorithm can have an exponential running time.

Critical Analysis of the Witness Algorithm

The experimental results presented in [5] suggests that witness algorithm does not scale to POMDPs with large state spaces. In practice the algorithm becomes impractical for a moderate size $|S| > 15$ and $|\Omega| > 15$ where $|S|$ denotes the number of states and $|\Omega|$ denotes the number of observations.

The witness algorithm unfortunately can take many iterations to find an approximately optimal value function and for problems with large number of observations the α vector set can grow explosively from iteration to iteration. Nonetheless it is often the case that a near optimal policy is obtained before Q values have converged to their optimal values. This means that one can actually terminate the value iteration process earlier and can still come up with excellent policies.

Critical Analysis Of Point Based Value Iteration

Since the PBVI starts with an initial set of belief B_0 and performs a number of backup stages and expands this set of belief to B_1 by sampling more beliefs and the process repeats itself with interleaving steps of value iteration and belief set expansion, it is a heuristic to let B_t cover a wide area of belief space. This comes at a cost as it requires computing the distance between all $b_t \in B$. By backing up all $b \in B_t$, the PBVI algorithm generates at each stage approximately $|B_t|$ vectors which

can lead to slow performance in domains requiring large B_t.

The algorithm performs one step to stochastically select belief points for each action-observation pair. One might find learning enhanced exploration and a bounding techniques in order to explicitly attempt to sample the reachable space.

The point based value iteration algorithm explicitly focuses on the curse of history, rather than the curse of dimensionality. Better techniques that require the derivation of bounds that relies on both the curse on dimensionality as well as the curse of history would help guide future algorithm design.

References

[1] R. Bellman, Dynamic Programming, Princeton University Press, 1957 MR 0090477

[2] E. J. Sondik. The Optimal Control of Partially Observable Markov Processes. PhD thesis, Stanford University,1971.

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

[4] Cassandra, Anthony R., Leslie Pack Kaelbling, and Michael L. Littman. "Acting optimally in partially observable stochastic domains." AAAI. Vol. 94. 1994.

[5] M.L. Littman, A.R. Cassandra and L.P. Kaelbling, Efficient dynamic-programming updates in partially observable Markov decision processes, Technical Report CS-95-19, Brown University, Providence, RI,1996

[6] Pineau, Joelle, Geoff Gordon, and Sebastian Thrun. "Point-based value iteration: An anytime algorithm for POMDPs." IJCAI. Vol. 3. 2003.

[7] Bonet, Blai. "An e-optimal grid-based algorithm for partially observable Markov decision processes." Proc. of the 19th Int. Conf. on Machine Learning (ICML-02). 2002.

[8] Zhou, Rong, and Eric A. Hansen. "An improved grid-based approximation algorithm for POMDPs." IJCAI. 2001.

[9] Pineau, Joelle, Geoffrey Gordon, and Sebastian Thrun. Tractable planning under uncertainty: exploiting structure. Diss. Carnegie Mellon University, the Robotics Institute, 2004.

[10] Smallwood, R. and Sondik, E.. 1973. The optimal control of partially observable Markov processes over a finite horizon. Operations Research. pp. 1071-1088.

[11] Astrom. K. J. 1965. Optimal control of Markov decision processes with incomplete state estimation. Journal of mathematical analysis and applications 10, 174-205

[12] Cassandra, Anthony, Michael L. Littman, and Nevin L. Zhang. "Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes." Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence. Morgan Kaufmann Publishers Inc., 1997.

[13] Zhang, Nevin Lianwen, and Weihong Zhang. "Speeding up the convergence of value iteration in partially observable Markov decision processes." Journal of Artificial Intelligence Research 14 (2001): 29-51.

[14] Kaelbling, Leslie Pack, Michael L. Littman, and Anthony R. Cassandra. "Planning and acting in partially observable stochastic domains." Artificial intelligence 101.1-2 (1998): 99-134.