McGill University

ECSE 506: Stochastic Control and Decision Theory

Semester: Winter 2020

Final Project Report

An Overview of Value Iteration Algorithms for POMDP

Instructor: Prof. Aditya Mahajan

Submitted by: Jilan Samiuddin

April 30, 2020

1 Introduction

In a Markov Decision Process (MDP), the agent has full observability of the states, however, that is not case for a Partially Observable MDP (POMDP), where the agent does not have a perfect vision of the environment; unlike MDP, probabilistic observations replace explicit state information. MDPs are easier to solve and therefore have been adapted in several industrial non-determinisic applications when the agents have perfect vision. This is due to the existence of efficient algorithms to solve problems with full observability, however, the computational complexity sharply rises when partial observability appears in the problem. This is a significant drawback that POMDPs suffer and the reason why they are not implemented in large-scale problems. A lot of research is undergoing in the machine-learning community trying to develop algorithms to solve POMDPs more efficiently since POMDPs encapsulates real-life systems much better than an MDP; a real-life system most likely will have uncertainties and the POMDP structure is developed to train an agent under uncertainties [1].

2 Brief overview of POMDP Structure

A POMDP is formally defined as the tuple $\langle S, A, T, R, Z, O, \gamma \rangle$, where,

- S is the finite set of states available in the environment for the agent to access,
- A is the finite set of actions available to the agent,
- $T : S \times A \rightarrow \Pi(S)$ is the set of conditional transition probabilities,
- $R : A \times S \rightarrow \mathbb{R}$ is the reward function,
- Z is the set of observations the agent experiences from the environment,
- $O : S \times A \rightarrow \Pi(Z)$ is the set of conditional observation probabilities which gives, for a given action and a resulting state, a probability distribution over all possible observations, and,
- γ is the discounting factor that reduces the utility of later rewards.

S, A, T, R are often called the often called the underlying MDP of the POMDP. In essence, in POMDPs the agent observes its environment based on its actions and the resulting states [1]. Note that the agent in a POMDP is Markovian with respect to the underlying state, however, because of deficiency in the sensor information, the agent is not Markovian with respect to the observations [2]. To make the agent Markovian in this regard as well, we use the agent's *belief* about its state given its history $h_t =< a_0, z_1, a_1, z_2, \ldots, a_{t-1}, z_t >$, where $a \in A$ and $z \in Z$:

$$b_t(s) = P(S_t = s | h_t), \qquad s \in S \tag{1}$$

2.1 Belief state

Belief is formally defined to be space of probability distributions over states, i.e., $\sum_{s \in S} b(s) = 1$. The belief state is independent of the policy determined by the agent and is also an information state, i.e., it is sufficient to predict itself and sufficient for performance evaluation [3]. The belief state provides

necessary statistics for the agent to optimize its policy [4]. Belief state can be updated at each time-step using Bayes rule as follows [5]:

$$\begin{aligned} b'(s') &= P(s'|z,a,b) \\ &= \frac{P(o|s',a,b)P(s'|a,b)}{P(z|a,b)} \\ &= \frac{P(z|s',a,b)\sum_{s\in S} P(s'|a,b,s)P(s|a,b)}{P(z|a,b)} \\ &= \frac{P(z|s',a)\sum_{s\in S} P(s'|a,s)P(s|b)}{P(z|a,b)} \\ &= \frac{O(s',a,z)\sum_{s\in S} T(s,a,s')b(s)}{P(z|a,b)}, \end{aligned} \tag{2}$$

where, the denominator, also sometimes denoted as $p_{z,a}$, is computed as follows:

$$\begin{aligned} p_{z,a} &= \sum_{s'\in S} P(z,s'|a,b) \\ &= \sum_{s'\in S} P(s'|a,b)P(z|s',a,b) \\ &= \sum_{s'\in S}\sum_{s\in S} P(s',s|a,b)P(z|s',a) \\ &= \sum_{s'\in S}\sum_{s\in S} P(s|a,b)P(s'|s,a,b)P(z|s',a) \\ &= \sum_{s'\in S} O(s',a,z)\sum_{s\in S} T(s,a,s')b(s). \end{aligned} \tag{3}$$

2.2 Value function of POMDP

Let us assume finite horizon for our POMDP. The agent needs to execute the best action from its current belief state, and to do so, it must have a policy π that will determine its actions to accumulate maximum reward. The policies map beliefs into actions

$$\pi : \mathcal{B} \rightarrow A.$$

The performance of the policy at a particular belief state then can be evaluated with the value function

$$V^{\pi}(b) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R(b,a)|b_0 = b\right], \tag{4}$$

where,

$$R(b,a) = \sum_{s\in S} b(s)R(s,a) \tag{5}$$

is similar to the reward function, but defined over b instead of s. $R(b,a)$ defines rewards for performing a in b. This allows us to define the optimal value function as (the recursive computational step)

$$V^*(b) = \max_{a \in A} \left[R(b,a) + \gamma \sum_{s' \in S} \underbrace{p_{z,a} V^*(b')}_{S(a,z)} \right]. \tag{6}$$

One significant property of the value function is that it is piecewise linear and convex (PWLC) for every horizon length [6]. This property is the cornerstone in solving POMDPs as it allows us to represent the value function by a finite number of linear segments. The function $S(a,z)$ is called the transformed value function [7]. Note that while $V(b)$ allows us to assess the performance of a policy if the agent has a belief b, $S(a,z)$ allows us to determine how good is the policy given the agent started from b, performed action a and observed z. $S(a,z)$ is also PWLC. The optimal policy is simply a follow-up of equation (6):

$$\pi^*(b) = \operatorname*{argmax}_{a \in A} \left[R(b,a) + \gamma \sum_{s' \in S} p_{z,a} V^*(b') \right]. \tag{7}$$

2.2.1 Alpha (α) vectors

Since $V(b)$ is PWLC, we can also define a finite set Γ of $|S|$-dimensional hyperplanes to represent it. These hyperplanes are also known as α-vectors represented by the equations' (of the hyperplanes) coefficients. This allows to write the optimal value function as follows:

$$V^*(b) = \max_{\alpha \in \Gamma_t} \alpha \cdot b, \tag{8}$$

where, $\alpha \cdot b = \sum_{s \in S} b(s)\alpha(s)$ is the standard inner product between α and b. But the issue with equation (8) is that as t grows, the total number of linear functions required to represent $V^*(b)$ also grows exponentially.

3 Value iteration in POMDP

Several algorithms use Value iteration (VI) to compute the value function, however, they differ in the optimization techniques they use,. Nevertheless, the heart of the algorithm and the complexity remains the same. That is why in this section the most elementary algorithm, the *enumeration algorithm* [6], is presented.

3.1 Enumeration Algorithm

The steps in the enumeration algorithm for each iteration are as follows :

1. Generate α-vectors representing immediate rewards, i.e., $\forall a \in A$, $\alpha^{a,*} \leftarrow R(s,a)$.
2. Generate $|A||Z||\Gamma_{t-1}|$ α-vectors as follows:

$$\alpha_i^{a,z} \leftarrow R(s,a) + \gamma \sum_{z \in Z} p_{z,a} \alpha_i'(b'), \qquad \forall \alpha_i' \in \Gamma_{t-1}, \forall a \in A, \forall z \in Z \tag{9}$$

   $\alpha_i^{a,z}$ is the expected reward when the agent follows policy governed by α_i' after performing a and observing z.

3. Regroup α-vectors found in steps 1 and 2:

$$\Gamma^{a,*} \leftarrow \cup_{a \in A} \{\alpha^{a,*}\} \tag{10}$$

$$\Gamma^{a,z} \leftarrow \cup_{a \in A, z \in Z} \{\alpha^{a,z}\} \tag{11}$$

4. Apply cross-sum operator over all observations:

$$\Gamma^{a} = \Gamma^{a,*} \oplus \Gamma^{a,z_1} \oplus \Gamma^{a,z_2} \oplus \ldots \tag{12}$$

5. The new Γ_t is created as follows:

$$\Gamma_t = \cup_{a \in A} \Gamma^{a} \tag{13}$$

   Here, Γ_t has all the α-vectors which form V_t.

3.2 Computational complexity per iteration

To understand the complexity involved, let $\mathcal{V}$ be a set of t-step policy tree p (see Figure 1) defined by a choice of action and one $(t-1)$-step policy for each possible observation. Here, action$(p) \in \mathcal{A}$ is the initial action and **choice**(p, z) is the $(t-1)$-step policy tree followed if z is observed after the first step. More precisely, it is a collection of a root node associated with an action a and $|Z|$ sub-trees each defining a $(t-1)$-step policy tree. The key idea in enumeration algorithm is to use $\mathcal{V}_{t-1}$, the set of *useful* $(t-1)$-step policy trees, to build a superset $\mathcal{V}_t^+$ of the useful t-step policy trees. At a particular belief state for any choice of policy sub-tree, there is always a useful sub-tree that is at least as good at that state, rendering non-useful policy sub-tree. In other words, many of the α-vectors are dominated[1] by the rest of the vectors allowing the dominated vectors to be removed without affecting equation (8). The enumeration algorithm always generates $\mathcal{V}_t^+$ which has $|\mathcal{A}||\mathcal{V}_{t-1}|^{|Z|}$ elements, i.e., it is exponential in $|Z|$. This is the reason why several algorithms use *pruning* that removes the dominated vectors allowing to speedup the next iteration. Pruning requires linear programming and does not add to the asymptotic complexity of the algorithm [1]. However, such exponential growth in complexity makes enumeration algorithm feasible only for small-scale problems as it makes the algorithm prohibitively expensive.

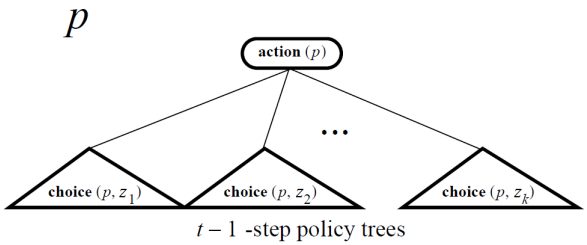

Figure 1: Policy tree [5]

[1] $\alpha \in \Gamma$ is dominated if $\exists \alpha' \in \Gamma$ such that $\alpha' \cdot b \geq \alpha \cdot b$, $\forall b \in \mathcal{B}$.

4 Numerical example using Enumeration Algorithm

In this section, a numerical example upto horizon two is shown; later horizons can be computed exactly the same way. One of the most common example problems in the POMDP realm is the tiger-problem in which there are two doors – behind one door is a tiger (with a penalty of -100) and a prize behind the other (a reward of $+10$). The agent does not know behind which door is the tiger and thus there are two states – the tiger is behind the left door or behind the right door represented by s_L and s_R respectively. The agent can either listen (a_l), open the left door (a_L) or open the right door (a_R). However, listening gives a noisy observation of the tiger's position, which is illustrated by the observation probability in Table 3(a) where z_L and z_R are observations of the tiger being on the left and right respectively. Note that the values in Table 3(a) are slightly modified from those found in most literature and tutorials; this can be interpreted as the agent has better hearing in his right ear. a_l is not free and costs -1. Immediate rewards, transition probabilities[2], and observation probabilities are listed in Table 1, Table 2 and Table 3 respectively. The objective of the agent is to maximize the reward by opening the door that has prize behind it. Note, for the simplicity of calculations, the discount factor $\gamma = 1$ is used in this section. The agent's belief state is $b = [b_{L(\text{eft})}, b_{R(\text{ight})}]$. We assume the initial belief state to be [0.5,0.5].

	s_L	s_R
a_l	-1	-1
a_L	-100	10
a_R	10	-100

Table 1: Immediate reward, $R(s,a)$

	s'_L	s'_R
s_L	1	0
s_R	0	1

(a) For action a_l

	s'_L	s'_R
s_L	0.5	0.5
s_R	0.5	0.5

(b) For action a_L

	s'_L	s'_R
s_L	0.5	0.5
s_R	0.5	0.5

(c) For action a_R

Table 2: Transition probabilities, $T(s,a,s') = P(s'|s,a)$

	z_L	z_R
s'_L	0.6	0.4
s'_R	0.2	0.8

(a) For action a_l

	z_L	z_R
s'_L	0.5	0.5
s'_R	0.5	0.5

(b) For action a_L

	z_L	z_R
s'_L	0.5	0.5
s'_R	0.5	0.5

(c) For action a_R

Table 3: Observation probabilities, $O(s',a,z) = P(z|s',a)$

Now, from equation (3), we can compute

$$p_{z_L,a_l} = 0.6(1\times0.5+0\times0.5)+0.2(0\times0.5+1\times0.5) = 0.4 \tag{14}$$

$$p_{z_R,a_l} = 0.4(1\times0.5+0\times0.5)+0.8(0\times0.5+1\times0.5) = 0.6 \tag{15}$$

Similarly, we can compute

[2] Action a_l does not change the state of the world. Actions a_L and a_R reset the problem.

$$p_{z_L,a_L} = p_{z_R,a_L} = p_{z_L,a_R} = p_{z_R,a_R} = 0.5 \tag{16}$$

The updated belief states using equation (2) and results from equations (14), (15) and (16) for different action-observation pairs are:

- For the pair (a_l, z_L),

$$b'(s'_L) = \frac{0.6(1\times0.5+0\times0.5)}{0.4} = 0.75, \qquad b'(s'_R) = 1-0.75 = 0.25 \tag{17}$$

- For (a_l, z_R),

$$b'(s'_L) = \frac{0.4(1\times0.5+0\times0.5)}{0.6} = 0.33, \qquad b'(s'_R) = 1-0.33 = 0.67 \tag{18}$$

- For all other pairs (a_L, z_L), (a_L, z_R), (a_R, z_L), and (a_R, z_R), similarly we can find $b'(s'_L) = b'(s'_R) = 0.5$.

Now, at $t = 1$,

$$R(b,a) = \begin{Bmatrix} -1b_L & -1b_R \\ -100b_L & +10b_R \\ 10b_L & -100b_R \end{Bmatrix} \tag{19}$$

The resulting value function is the maximum of the three functions at each point (see Figure 2[3]):

$$V_1(b) = \max \begin{Bmatrix} -1b_L & -1b_R \\ -100b_L & +10b_R \\ 10b_L & -100b_R \end{Bmatrix} \tag{20}$$

Thus, the optimal policy at $t = 1$:

$$\pi_1(b) = \begin{cases} a_L, & \text{if } b_R < 0.1 \\ a_l, & \text{if } 0.1 \le b_R \le 0.9 \\ a_R, & \text{if } b_R > 0.9 \end{cases} \tag{21}$$

Using $V_1(b)$, the optimal action with $b_1 = [0.5, 0.5]$ is a_l. The number of α-vectors here is 3. However, because there is no dominated vector, no pruning is done.

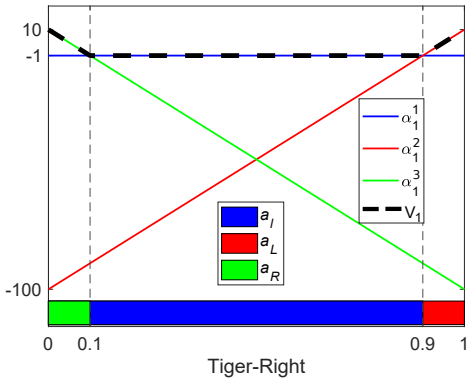

Figure 2: Value function at $t = 1$

[3]On the x-axis, 0 means the agent is certain of s_L and 1 means the agent is certain of s_R.

Now, for $t = 2$, at first using equation (2) and results from equations (14), (15) and (16), we get

$$V(b'|a_l,z_L) = \begin{Bmatrix} -1\left(\frac{0.6}{0.4}\right)b_L & -1\left(\frac{0.2}{0.4}\right)b_R \\ -100\left(\frac{0.6}{0.4}\right)b_L & +10\left(\frac{0.2}{0.4}\right)b_R \\ 10\left(\frac{0.6}{0.4}\right)b_L & -100\left(\frac{0.2}{0.4}\right)b_R \end{Bmatrix} = \frac{1}{0.4}\begin{Bmatrix} -0.6b_L & -0.2b_R \\ -60b_L & +2b_R \\ 6b_L & -20b_R \end{Bmatrix} \tag{22}$$

$$V(b'|a_l,z_R) = \begin{Bmatrix} -1\left(\frac{0.4}{0.6}\right)b_L & -1\left(\frac{0.8}{0.6}\right)b_R \\ -100\left(\frac{0.4}{0.6}\right)b_L & +10\left(\frac{0.8}{0.6}\right)b_R \\ 10\left(\frac{0.4}{0.6}\right)b_L & -100\left(\frac{0.8}{0.6}\right)b_R \end{Bmatrix} = \frac{1}{0.6}\begin{Bmatrix} -0.4b_L & -0.8b_R \\ -40b_L & +8b_R \\ 4b_L & -80b_R \end{Bmatrix} \tag{23}$$

Similarly,

$$V(b'|a_L,z_L) = V(b'|a_L,z_R) = V(b'|a_R,z_L) = V(b'|a_R,z_R) = \frac{1}{0.5}\begin{Bmatrix} -0.5b_L & -0.5b_R \\ -50b_L & +5b_R \\ 5b_L & -50b_R \end{Bmatrix} \tag{24}$$

Thus, the transformed value functions are (from equation (6)):

$$S(a_l,z_L) = 0.4\left[\frac{1}{0.4}\begin{Bmatrix} -0.6b_L & -0.2b_R \\ -60b_L & +2b_R \\ 6b_L & -20b_R \end{Bmatrix}\right] = \begin{Bmatrix} -0.6b_L & -0.2b_R \\ -60b_L & +2b_R \\ 6b_L & -20b_R \end{Bmatrix} \tag{25}$$

$$S(a_l,z_R) = 0.6\left[\frac{1}{0.6}\begin{Bmatrix} -0.4b_L & -0.8b_R \\ -40b_L & +8b_R \\ 4b_L & -80b_R \end{Bmatrix}\right] = \begin{Bmatrix} -0.4b_L & -0.8b_R \\ -40b_L & +8b_R \\ 4b_L & -80b_R \end{Bmatrix} \tag{26}$$

$$S(a_L,z_L) = S(a_L,z_R) = S(a_R,z_L) = S(a_R,z_R) = 0.5\left[\frac{1}{0.5}\begin{Bmatrix} -0.5b_L & -0.5b_R \\ -50b_L & +5b_R \\ 5b_L & -50b_R \end{Bmatrix}\right] = \begin{Bmatrix} -0.5b_L & -0.5b_R \\ -50b_L & +5b_R \\ 5b_L & -50b_R \end{Bmatrix} \tag{27}$$

The results are intuitive (see Figure 3) – e.g., if the agent observes z_L after action a_l, his belief about s_L expands and thus more likely to perform a_R at $t = 2$ than at $t = 1$.

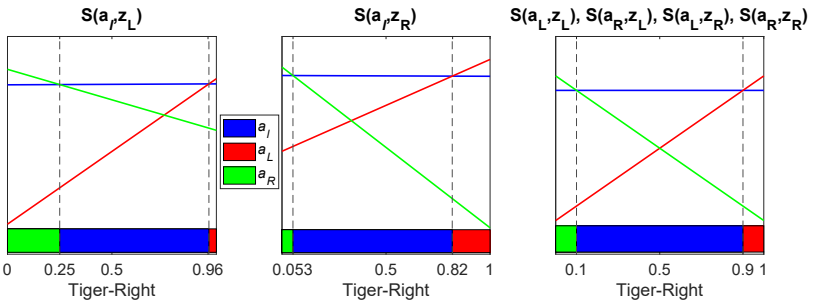

Figure 3: Transformed value functions $S(a,z)$

Now, using equations (25), (26) and (27), the expected values after observations are

$$\bar{V}(b'|a_l) = \max \begin{Bmatrix} -0.6b_L & -0.2b_R \\ -60b_L & +2b_R \\ 6b_L & -20b_R \end{Bmatrix} + \max \begin{Bmatrix} -0.4b_L & -0.8b_R \\ -40b_L & +8b_R \\ 4b_L & -80b_R \end{Bmatrix}$$

$$\overset{(a)}{=} \max \begin{Bmatrix} -1b_L & -1b_R \\ -40.6b_L & +7.8b_R \\ 3.4b_L & -80.2b_R \\ -60.4b_L & +1.2b_R \\ -100b_L & +10b_R \\ -56b_L & -78b_R \\ 5.6b_L & -20.8b_R \\ -34b_L & -12b_R \\ 10b_L & -100b_R \end{Bmatrix} \overset{(b)}{=} \max \begin{Bmatrix} -1b_L & -1b_R \\ -40.6b_L & +7.8b_R \\ -100b_L & +10b_R \\ 5.6b_L & -20.8b_R \\ 10b_L & -100b_R \end{Bmatrix} \tag{28}$$

where, (a) is from the nine possible combinations between the two max terms, and (b) is from pruning. We can visualize the need of pruning from Figure (4). There were initially nine α-vectors, however, four of them were dominated by the remaining five vectors, and thus were pruned.

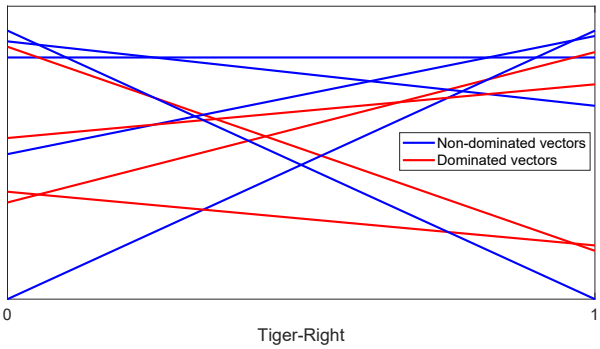

Figure 4: α-vectors from equation (28) before pruning

Similarly, we have

$$\bar{V}(b'|a_L) = \bar{V}(b'|a_R) = \max \begin{Bmatrix} -0.5b_L & -0.5b_R \\ -50b_L & +5b_R \\ 5b_L & -50b_R \end{Bmatrix} + \max \begin{Bmatrix} -0.5b_L & -0.5b_R \\ -50b_L & +5b_R \\ 5b_L & -50b_R \end{Bmatrix}$$

$$= \max \left\{ \begin{pmatrix} -1b_L & -1b_R \\ -50.5b_L & +4.5b_R \\ 4.5b_L & -50.5b_R \\ -50.5b_L & +4.5b_R \\ -100b_L & +10b_R \\ -45b_L & -45b_R \\ 4.5b_L & -50.5b_R \\ -45b_L & -45b_R \\ 10b_L & -100b_R \end{pmatrix} \right\} \overset{\text{(pruning)}}{=} \max \left\{ \begin{pmatrix} -1b_L & -1b_R \\ -100b_L & +10b_R \\ 10b_L & -100b_R \end{pmatrix} \right\} \tag{29}$$

Finally, it should be noted that since the optimal action for $t = 1$ was a_l, the immediate reward under both the observations is -1. Thus, using results from equations (28) and (29),

$$V_2(b) = \max \left(\begin{bmatrix} -1b_L & -1b_R \end{bmatrix}_{9\times 1} + \left\{ \begin{pmatrix} -1b_L & -1b_R \\ -40.6b_L & +7.8b_R \\ -100b_L & +10b_R \\ 5.6b_L & -20.8b_R \\ 10b_L & -100b_R \\ -1b_L & -1b_R \\ -100b_L & +10b_R \\ 10b_L & -100b_R \\ -1b_L & -1b_R \\ -100b_L & +10b_R \\ 10b_L & -100b_R \end{pmatrix} \right\} \right) \overset{\text{(pruning)}}{=} \max \left\{ \begin{pmatrix} -2b_L & -2b_R \\ -41.6b_L & +6.8b_R \\ -101b_L & +9b_R \\ 4.6b_L & -21.8b_R \\ 9b_L & -101b_R \end{pmatrix} \right\} \tag{30}$$

The value function from equation (30)[4] is illustrated in Figure 5. There are five strategies for the agent to follow represented by five different colored bars at the bottom. These five strategies are illustrated in Figure 6. After taking the optimal action a_l at $t = 1$[5], if the agent observes z_L for which his updated belief state is [0.75,0.25] (equation (17)), his optimal action will be a_l, and, if he observes z_R for which his updated belief state is [0.33,0.67] (equation (18)), his optimal action still will be a_l.

[4]The value function has been verified by running the POMDP model in both R [8] and Julia [9].

[5]Recall that initial belief state is [0.5,0.5] for which the optimal action was a_l using equation (21).

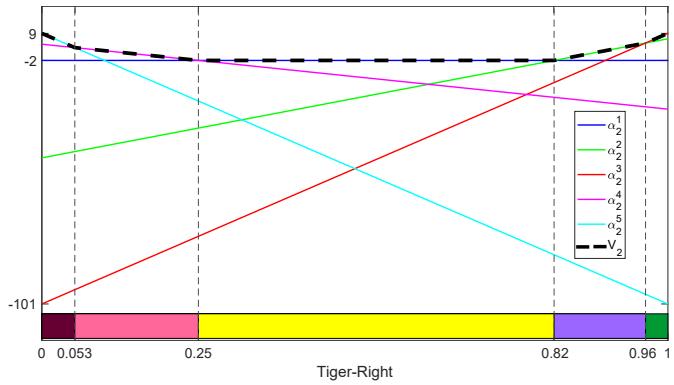

Figure 5: Value function at $t = 2$

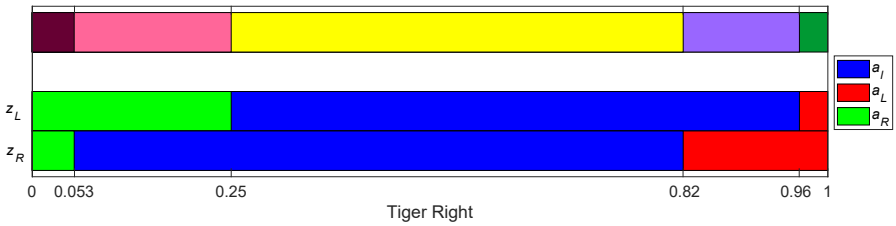

Figure 6: Strategy partition at $t = 2$

5 Value Iteration Algorithms

In this section, few of the many existing algorithms for performing VI in POMDP problems will be discussed.

5.1 Linear Support Algorithm

Notation Cheng's thesis: X – state, Y,d – action, D – action space, Observation – Z,θ, q – observation probability, I – history

The linear support algorithm [10] uses the idea of enumeration algorithm but with lesser strict constraints. The non-dominated vectors *support* the value function and thus the name linear support. Let R_{α_i} be a non-empty belief space region called the support region in which α_i is the non-dominated optimal vector. If a belief state b in R_{α_i} can testify that α_i is non-dominated, then it is called a *witness*. In Figure 7a, b is a witness for α_2 in its support region R_{α_2}.

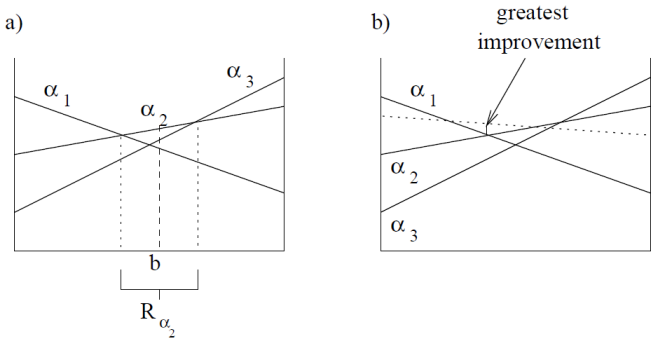

Figure 7: Example of a support region [11]

The key idea in linear support algorithm is that if there is an error between the approximate value function and the ideal value function, the biggest difference will occur at a vertex. Thus, witness points are sought at the vertices of the approximate value function. To demonstrate this, when a dotted α-vector is added to the existing α-vectors as shown in Figure 7b, the value function improves in at least one belief state corresponding to a vertex, and moreover, the greatest improvement surely happens at one of the vertices (intersection of α_1 and α_2 in this particular case).
The dynamic programming in linear support algorithm finds vertex that witnesses the absence of the a support vector providing the greatest improvement. When the witness is identified, it generates only the support vector. With two states problems, these corners can be located efficiently, however, for higher dimensions, the number of vertices is often exponential in dimensionality.

5.2 Witness Algorithm

As described in section 3.2, V_{t-1} and V_t can be represented using collections of policy trees $\mathcal{V}_t$ and $\mathcal{V}_{t-1}$ respectively. The witness algorithm [5] computes the set Q_t^a for each action, where

$$Q_t^a(b) = \sum_{s \in S} b(s) R(s,a) + \gamma \sum_{z \in Z} p_{z,a} V_{t-1}(b') \tag{31}$$

provides the expected reward for performing a from belief state b and then continue to act optimally for the remaining $(t-1)$ steps. These Q-functionsare PWLC as well and thus we can represent them using a unique minimal useful set of policy tree. The value function is then given by

$$V_t(b) = \max_{a \in A} Q_t^a(b) \tag{32}$$

The objective of the algorithm is to find for each action a the minimal set of policy trees to represent Q_t^a. To do that, consider U_a the ideal set of policy trees which represents the optimal value function V_t. U_a is initialized with a single policy tree with respect to action a at the root node and then updated incrementally to include only a set of optimal policy trees from Q_t^a. After computing Q_t^a from equation (31) and $\tilde{Q}_t^a$ from U_a, the algorithm searches for a belief state b (also called a *witness*) such that $Q_t^a > \tilde{Q}_t^a$. If a witness exists, a policy tree is constructed with action a at the root that yields the best value at b and adds it to U_a. The process is repeated until the algorithm finds no more witness points that proves the current Q-function is perfect (or at least near-perfect defined by a tolerance). This can be stated mathematically as: the process is repeated until for some tolerance $\varepsilon > 0$,

$$|V_t(b) - V_{t-1}(b)| < \varepsilon, \quad \forall b. \tag{33}$$

To understand how to find the witness point, let p be a t-step policy tree and p' be a $(t-1)$-step policy tree. We can then define p_{new} as a $(t-1)$-step policy tree that coincides with p in its action and all its sub-trees except for observation z_i. Now, for any b and $\tilde{p} \in U_a$, $V_{p_{\text{new}}}(b) - V_{\tilde{p}}(b)$ quantifies the benefit of of following p_{new} instead of $\tilde{p}$. Also, let δ be the minimum amount of improvement of p_{new} over any policy tree in U_a, i.e., δ is a bound on the difference $V_{p_{\text{new}}}(b) - V_{\tilde{p}}(b)$. Algorithm 1 [1] is defined to maximize the benefit of following p_{new} instead of $\tilde{p}$, i.e., if it finds $\delta \geq 0$, then p_{new} is an improvement over all $\tilde{p} \in U_a$ and thus b is a witness point. It is easy to notice that Algorithm 1 is a simple linear programming problem.

Algorithm 1 Linear program to find witness point

Inputs:

U_a, p_{new}

Variables:

δ, $b(s)$ for each $s \in S$

Maximize: δ

Improvement constraints:

For each $\tilde{p} \in U_a$: $V_{p_{\text{new}}}(b) - V_{\tilde{p}}(b) \geq \delta$

Simplex constraints:

For each $s \in S$: $b(s) \geq 0$

$\sum_{s \in S} b(s) = 1$

5.3 Point-based Value Iteration

The introduction of point-based value iteration (PBVI) algorithm [12] allowed to approximately solve large-scale POMDPs speedily compared to the classical ones. The speed comes from the fact that PBVI updates the value function for a limited set of belief $B = \{b_0, b_1, \ldots, b_q\}$ unlike exact algorithms, and thus instead of generating $|A||-_{t-1}|^{|Z|}$ new α-vectors, it generates $|B|$ at most. However, the speed comes at the cost of accuracy due to the discrete choice of belief states. The intuition behind PBVI is that the policy for a belief state b is likely to perform very good at a nearby belief state b'. The nearness can be computed using the euclidean distance[6]:

$$\text{dist}(b, b') = \sqrt{\sum_{s \in S} [b(s) - b'(s)]^2} \tag{34}$$

The PBVI algorithm maintains a full α-vector for each belief point and thus ensures PWLC of the value function (see Figure 8).

PBVI is an *anytime* algorithm that alternates between steps of value iteration and steps of belief set expansion offering a trade-off between computational time and accuracy. The basic idea is that it initializes a set B and applies point-based value backup operations. Then it expands the set and finds a new solution for the new set.

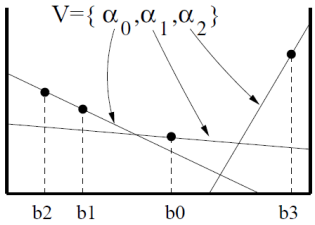

Figure 8: POMDP value function representation using PBVI [12]

5.3.1 Point-based value backup

The point-based update is done using the following steps:

1. Generate α-vectors representing immediate rewards, i.e., $\forall a \in A$, $\Gamma^{a,*} \leftarrow \alpha^{a,*} = R(s,a)$.

[6]The norm type does not seem to affect the results in practice [12].

2. Generate $|A||Z||\Gamma_{t-1}|$ α-vectors $\forall \alpha_i' \in \Gamma_{t-1}, \forall a \in A, \forall z \in Z$:

$$\Gamma^{a,z} \leftarrow \alpha_i^{a,z} = \gamma \sum_{z \in Z} p_{z,a} \alpha_i'(b') \tag{35}$$

3. Operations over finite set of points allows the cross-sum to be simplified to

$$\Gamma_b^a = \Gamma^{a,*} + \sum_{z \in Z} \underset{\alpha \in \Gamma^{a,z}}{\operatorname{argmax}} (\alpha \cdot b) \tag{36}$$

4. Find the best action for individual belief points

$$V \leftarrow \underset{\Gamma_b^a, \forall a \in A}{\operatorname{argmax}} (\Gamma_b^a \cdot b), \quad \forall b \in B \tag{37}$$

where V has $|B|$ components, i.e., the size remains constant.

The point-based update renders pruning of α-vectors unnecessary due to the fact that the size of V is constant, however, in the case where two (or more) nearby belief points support the same vector (see b_1 and b_2 in Figure 8), only then pruning is done to avoid adding of the same vector twice (or more). For finite horizon T, the backup operation is done T times. For infinite horizon, we select the horizon T such that

$$\left(\underbrace{\max_{a,s} R(s,a)}_{R_{\max}} - \underbrace{\min_{a,s} R(s,a)}_{R_{\min}} \right) \gamma^T < \varepsilon \tag{38}$$

where $\varepsilon > 0$ is some tolerance value.

5.3.2 Belief point set expansion

For a given $b \in B$, PBVI stochastically simulates an action by first sampling a state s from the distribution b, then sampling an observation using $O(s,a,z)$ and finally computing $b_a = \{b_{a_0}, b_{a_1}, \ldots\}$ using equation (2). It then measures distance between b_{a_i} and B using equation (34) and discards b_{a_i} if $b_{a_i} \in B$. Lastly, it retains b_{a_i} that is farthest from any point in B.

5.3.3 Convergence and error bounds

Let V_n^B be the value function estimated by PBVI for any belief set B and horizon n, and V_n^* be the true value function. As $n \to \infty$, V_n^B does not necessarily converge, but there are error bounds that it still holds (see Lemma 1 and Theorem 1 in [12] for proofs):

1. Let b' be the point where PBVI makes its worst pruning error and b be the closest sampled belief to b'. The error introduced by PBVI's pruning step is at most

$$\eta_{\text{prune}} = \frac{(R_{\max} - R_{\min})\epsilon_B}{(1-\gamma)} \tag{39}$$

where, $\epsilon_B = \max_{b'} \min_{b} \operatorname{dist}(b, b')$.

2. For any belief set B and any horizon n, the error of the PBVI algorithm $\eta_n = \left\|V_n^B - V_n^*\right\|_\infty$ is bounded by

$$\eta_n = \frac{(R_{\max} - R_{\min})\epsilon_B}{(1-\gamma)^2} \tag{40}$$

5.4 Comparative analysis

Some of the differences between the algorithms studied in sections 5.1, 5.2 and 5.3 are discussed in this section.

1. In linear support algorithm, witness points are sought at vertices, whereas, in witness algorithm, witness points are sought in the whole belief space. PBVI, on the other hand, does not require witness points.

2. Both linear support algorithm and witness algorithm compute exact value function, however, PBVI finds an approximate of the value function because of its sampling of the belief state.

3. PBVI algorithm has a tradeoff between accuracy and speed, but the other two does not sacrifice accuracy.

4. Updates of linear support algorithm and witness algorithm are exponential in time, but PBVI updates in polynomial time. It is worth noting that it is possible to create families of POMDPs that linear support algorithm solves in polynomial time which witness algorithm solves in exponential time [1]. These are problems with S and $\mathcal{V}_t$ are very small and Q_t^a is exponentially larger for some action a.

5. Pruning is essential for witness algorithm and linear support algorithm, whereas, in PBVI is only done when there are repeating α-vectors. This allows PBVI to save computational time by avoiding linear programming for pruning.

6. Linear support algorithm and witness algorithm are suitable only for small-scale problems (around a dozen states), but PBVI can handle large-scale problems.

7. In [13], it was shown that the run-time for witness algorithm was significantly higher for the Tiger-problem in contrast to PBVI. Linear support algorithm having the similar philosophy of solving a POMDP, can be expected to demonstrate similar run-time as witness algorithm.

5.5 Critical analysis

Linear support algorithm is very fast when number of states are small, but scales poorly since number of vertices can grow exponentially with the size of the state space. Visiting each vertex is just one of the algorithm linear support algorithm performs, and in worst-case scenarios, the run-time can be expected to grow at least exponentially in the size of the one-stage POMDP problem.
Witness algorithm has been used to solve POMDPs with up to 16 states which is considerably larger than prior state of the art, however, it is not efficient to solve for larger problems. Witness algorithm can take significant number of iterations to find the optimal value function, and for large number of observations

can become intractable. Although, often a near-optimal policy is achieved long before convergence and thus halting the process early can still yield excellent policies [14].

A critical aspect of PBVI is to perform backups on the right belief states. It is important to update belief states that would be likely to be visited by the optimal policy; selecting large number of belief states will tackle this issue, but at the expense of speed. The growth in the number of selected belief points slows down the performance of value update stage. The same influence comes from the number of backups. Domains that require large B_t slows down PBVI since at each stage it generates $|B_t|$ vectors by backing all $b \in B_t$. PBVI handles the curse of history pretty well, however, the curse of dimensionality is yet to be tapped by PBVI.

For all the algorithms discussed, one aspect is common to all of them: they are all offline algorithms. Offline algorithms require too much resources to generate decent solutions for large-scale problems, e.g., in a game of chess there are 10^{43} states and it is not feasible to plan policies offline for each and every possible scenarios. However, an online algorithm, on the other hand, requires lesser resources and makes decisions on the go as it asses the scenario real-time.

6 Conclusion

The objective of this report is to introduce briefly few of the existing algorithms to solve POMDPs using value iteration and the computational complexity involved in solving them. Using Sondik's Enumeration algorithm, the simplest of value iteration techniques for POMDPs, the complexity that arise due to generation of α-vectors in each horizon was shown. A numerical example was also provided to illustrate this. At $t = 2$, there were a total of $(3 \times 3^2 =)$ 27 α-vectors (see equations (28) and (29) before pruning step), including both dominated and non-dominated vectors, of which 22 were pruned. A POMDP problem with higher dimension is surely to become intractable. Three different value iteration algorithms are then briefly discussed. Both linear support algorithm and the witness algorithm are able to solve small-scaled POMDPs but PBVI can handle larger sized problems much more computational efficiently at the cost of some accuracy. A run-time test between the algorithms could not be carried out because the linear support algorithm requires external linear programming solver to be integrated in Julia [9] and due to time constraint, it could not be implemented. In R [8], the linear support algorithm is not made available, possibly because of the mentioned reason. Finally, a comparative analysis and critical analysis between the three algorithms were provided.

References

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

[2] Guy Shani, Joelle Pineau, and Robert Kaplow. A survey of point-based pomdp solvers. *Autonomous Agents and Multi-Agent Systems*, 27(1):1–51, 2013.

[3] Aditya Mahajan. Theory: Basic model of a pomdp, ecse 506, mcgill university. https://adityam.github.io/stochastic-control/pomdp/pomdp/, Winter 2020. Last Accessed: 2020-04-19.

[4] Karl J Astrom. Optimal control of markov decision processes with incomplete state estimation. *J. Math. Anal. Applic.*, 10:174–205, 1965.

[5] Michael L Littman. The witness algorithm: Solving partially observable markov decision processes. *Brown University, Providence, RI*, 1994.

[6] Edward Jay Sondik. The optimal control of partially observable markov processes. Technical report, Stanford Univ Calif Stanford Electronics Labs, 1971.

[7] Pomdp value iteration example. https://www.pomdp.org/. Last Accessed: 2020-04-22.

[8] Hossein Kamalzadeh and Michael Hahsler. Pomdp: Introduction to partially observable markov decision processes. *Tekn. rapport*, 2019.

[9] Maxim Egorov, Zachary N. Sunberg, Edward Balaban, Tim A. Wheeler, Jayesh K. Gupta, and Mykel J. Kochenderfer. POMDPs.jl: A framework for sequential decision making under uncertainty. *Journal of Machine Learning Research*, 18(26):1–5, 2017.

[10] Hsien-Te Cheng. *Algorithms for partially observable Markov decision processes*. PhD thesis, University of British Columbia, 1988.

[11] Pascal Poupart. *Exploiting structure to efficiently solve large scale partially observable Markov decision processes*. PhD thesis, University of Toronto, 2005.

[12] Joelle Pineau, Geoff Gordon, Sebastian Thrun, et al. Point-based value iteration: An anytime algorithm for pomdps. In *IJCAI*, volume 3, pages 1025–1032, 2003.

[13] Fan Sun. An empirical comparison of various representations of dynamic systems. 2010.

[14] Michael L Littman, Anthony R Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up. In *Machine Learning Proceedings 1995*, pages 362–370. Elsevier, 1995.