Randomized POMDP Planning Algorithms

Mohammad Taufeeque
taufeeque@cse.iitb.ac.in

Shivaram Kalyanakrishnan
shivaram@cse.iitb.ac.in

IIT Bombay

May, 2021

Contents

1 Introduction

POMDPs are capable of modelling a large class of decision and planning problems. However, solving large POMDPs optimally is infeasible. The following text reports observations of various experiments related to solving large POMDPs using various strategies conducted as part of an R&D project at IIT Bombay.

The section 2 and 3 in the text formally define MDPs and POMDPs and also mention how POMDP solutions (formally called policies) are represented. Important existing POMDP solving algorithms relevant to the experiments conducted are summarized in section 4. Section 5 describes the Subset Update algorithm introduced in [8]. Section 6 mentions a way to reduce the FSC sizes during each iteration by a logical pruning step. Section 7 specifies a few methods to combine two FSCs into a single FSC. Section 8 reports certain empirical results. Some observations and possibilities of future work have been highlighted in the last section.

2 Sequential Decision Making Processes

A Sequential Decision-Making process involves an agent, interacting with its uncertain environment. At each time-step or horizon, the agent has to take an action based on the information it has amassed, so as to achieve a pre-decided goal.

2.1 MDPs

Markov Decision Processes (MDPs) are a class of commonly used formal models to represent uncertain but fully observable environments. An MDP is a 6-tuple $(S, A, T, \mathcal{K}, s_0, \gamma)$. Where,

- S is a finite set of states with $s_0 \in S$ as the agent's initial state
- A is a finite set of actions
- $T : S \times A \times S \rightarrow R$ is the transition function such that $\forall s \in S, a \in A : T(s, a, .)$ is a probability distribution over S
- $\mathcal{K} : S \times A \times S \rightarrow \mathbb{R}$ is the reward function, with γ as the discount factor

At each time-step t, the agent has to choose an action a_t from A, which makes it randomly change its state from $s_t \in S$ to $s_{t+1} \in S$, according to the pdf $T(s_t, a_t, .)$. The agent thus gets a reward of $\mathcal{K}(s_t, a_t, s_{t+1})$ for this time-step. It is assumed that the agent starts in the state s_0. The objective is to maximize the expected

value of the expression:

$$\sum_{t=0}^{h} \gamma^t \mathcal{K}(s_t, a_t, s_{t+1}) \tag{1}$$

The above expression is also called the finite horizon reward. When the horizon $h \to \infty$, it is called the infinite horizon reward. Throughout this text, we are concerned with the infinite horizon expected reward maximization problem.

2.2 POMDPs

Partially Observable Markov Decision Processes or POMDPs, extend MDPs to model partially observable environments. In the POMDP model, although the agent follows the transition function T, it cannot directly know which state it is in; it has to rely on the observations it gets. The inference of the state is made from the observation function $Z : S \times A \times O \to \mathbb{R}$, such that $\forall s \in S, a \in A : Z(s, a, .)$ is a probability distribution over O. $Z(s, a, o)$ is the probability that the agent receives the observation o, given that it just took the action a to reach state s. Thus, when the agent takes an action a_t to move to a state s_{t+1}, it sees an observation o_{t+1} based on the pdf $Z(s_{t+1}, a_t, .)$. The goal remains the same - maximize the infinite horizon expected reward:

$$\sum_{t=0}^{h} \gamma^t \mathcal{K}(s_t, a_t, s_{t+1}) \tag{2}$$

Since it is not known which state the agent is present in, a pdf over the states is maintained, called the belief vector or belief state, b. In each time-step, the vector b is updated based on the observation received. It is assumed that the initial belief state b_0 is known. Thus, a POMDP is completely defined as a 8-tuple $(S, A, T, O, Z, \mathcal{K}, b_0, \gamma)$. The belief state is also represented as a vector of size $|S| - 1$ representing the probabilities of the first $|S| - 1$ states. The belief space $\mathcal{B} = [0, 1]^{|S|-1}$ is a real-valued set in the $(|S| - 1)$-dimensional plane. $\mathcal{B}$ represents all possible belief states.

3 Policy Representations

A (PO)MDP policy is a function which specifies the action that has to be taken by the agent at each time-step, in all possible scenarios. It could include the amassed environmental information in various forms to specify the action for a time-step. We are specifically interested in a special type of policy, called the optimal infinite horizon policy. For the purpose of this text, it is a policy which when followed by the agent, maximizes the expected infinite horizon reward (2).

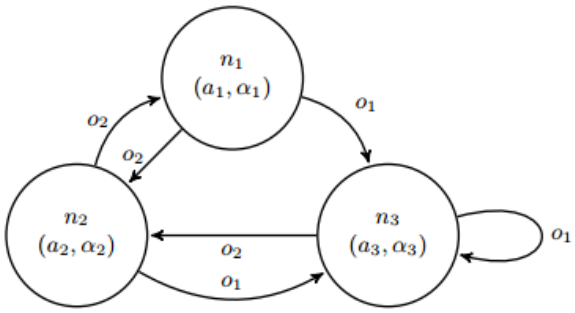

Figure 1: A sample FSC with $\mathcal{N} = \{n_1, n_2, n_3\}$. $\psi(n_i) = a_i, \forall i \in \{1, 2, 3\}$. Each node's successors for observations in $O = \{o_1, o_2\}$ are denoted by edges. α_i is the α-vector corresponding to node $n_i \; \forall i \in \{1, 2, 3\}$.

3.1 Finite State Contollers

A POMDP policy π is often represented as a Finite State Controller (FSC). An FSC or policy graph π is a triple $(\mathcal{N}, \psi, \eta)$ where:

- $\mathcal{N}$ is a set of controller nodes n, also known as internal memory states.
- $\psi : \mathcal{N} \rightarrow A$ is the action selection function that for each node n prescribes an action $\psi(n)$
- $\eta : \mathcal{N} \times O \rightarrow \mathcal{N}$ is the node transition function that for each node and observation assigns a successor node n'. $\eta(n, .)$ is essentially an observation strategy for the node n

Each node n is associated with a vector of length $|S|$, called an α-vector. For any α-vector α_i, $\alpha_i(s_j)$ is the expected infinite horizon reward that the agent will get, if it is currently in state s_j and starts following the policy π from α_i. A policy is also sometimes defined by simply defining a set of α-vectors, $\mathcal{V}$. Thus, the set $\mathcal{V}^\pi$ represents policy π. For the purpose of this text, an α-vector is always associated with an action and a observation strategy η. Thus, a α-vector defines a FSC node and vice versa. The two terms have been used interchangeably throughout the text. When an agent follows a policy, it first chooses an initial node or α-vector (say, n_0 or α_{n_0}). At each time-step t, it takes the action $a_t = \psi(n_t)$ associated with the current node and would receive an observation o_{t+1}. It then changes the current node to $n_{t+1} = \eta(n_t; o_{t+1})$, before the next time-step, t+1. Thus, for a given belief state b, the FSC promises a expected reward $R(b)$ of:

$$R(b) = \max_{n \in \mathcal{N}} [b \cdot \alpha_n] \tag{3}$$

Here, function $R : \mathcal{B} \rightarrow \mathbb{R}$ maps the belief states to their expected reward according to the FSC. R is also called the expected reward function. Also note, b is a vector with size equal to $|S|$. Its dot product has been taken with α_n, the α-vector corresponding to node n. Figure 1 shows a sample 3 node FSC.

3.2 Value Function

$V^{\pi} : \mathcal{N} \times S \rightarrow \mathbb{R}$, the value function for a policy π, specifies the value of each α-vector in $\mathcal{V}^{\pi}$. Thus,

$$V^{\pi}(n, s) = \alpha_n(s) \tag{4}$$

4 POMDP solving algorithms

In the infinite horizon case, an exact POMDP solving algorithm returns an optimal infinite horizon policy for an input POMDP. The following subsections describe some of the POMDP planning algorithms relevant to the experiments reported in this text.

4.1 Point Based Value Iteration

Instead of planning on the entire belief simplex $\mathcal{B}$, as exact value iteration does, point-based algorithms (Pineau et al. [9] and Spaan and Vlassis [12]) alleviate the computational load by planning only on a finite set of belief points $\hat{B}$. They utilize the fact that most practical POMDP problems assume an initial belief b_0, and concentrate planning resources on regions of the simplex that are reachable (in simulation) from b_0. Based on this idea, Pineau et al. [9] proposed a PBVI algorithm that first collects a finite set of belief points $\hat{B}$ by forward simulating the POMDP model and then maintains a single α-vector for each $b \in \hat{B}$. This is summarized in Algorithm 1.

Algorithm 1: Point based backup

Function `backup` $(\Gamma, \hat{B})$
- **Input :** Set of α-vectors Γ and Belief set $\hat{B}$
- **Output:** Updated set of α-vectors Γ'
- **foreach** $b \in \hat{B}$ **do**
  - $\alpha_a^o \leftarrow argmax_{\alpha \in \Gamma}\ \alpha \cdot b_a^o$, for every $a \in A, o \in O$
  - $\alpha_a(s) \leftarrow R(s,a) + \gamma \sum_{o,s'} T(s,a,s') Z(s',a,o) \alpha_a^o(s')$
  - $\alpha' \leftarrow argmax_{\{\alpha_a\}_{a \in A}} \alpha_a \cdot b$
  - **if** $\alpha' \notin \Gamma'$ **then**
    - $\Gamma' \leftarrow \Gamma' + \alpha'$
- **return** Γ'

4.2 Policy Iteration

Unless a value iteration algorithm converges for a given POMDP, the set of vectors obtained in the value iteration algorithm in each iteration represent finite horizon policies. Thus, it might not be appropriate to

use them for infinite horizon problems. Policy iteration on the other hand generates an infinite horizon policy in each iteration. Each subsequent policy being "better" than its predecessor. Thus, the algorithm can be stopped at any iteration and the current policy can be used. Hansen's policy iteration [3] and [5], is an iterative algorithm which repeatedly performs two steps: Policy Evaluation and Policy Improvement, until convergence. The algorithm 2 takes as input an initial FSC (eg: a cyclic 1-node FSC) and each iteration strictly improves the policy represented by the FSC. An FSC π_0 is a strict improvement of FSC π iff:

$$(\forall b \in \mathcal{B} \quad R'(b) \geq R(b)) \wedge (\exists b' \in \mathcal{B} \quad R'(b') > R(b')) \tag{5}$$

where R and R' are the expected reward funtions for π and π' respectively.

Each FSC node corresponds to an α-vector in a piecewise-linear and convex reward function. For a node n, $\psi(n)$ outputs the action associated with the node n, and $\eta(n, o)$ is the successor node of n after receiving observation o.

Policy Evaluation

The value function V^π of a FSC π, is calculated as follows:

$$V^\pi(n, s) = \sum_{s' \in S} T(s, \psi(n), s') R(s, \psi(n), s') + \gamma \sum_{s' \in S} \sum_{o \in O} T(s, \psi(n), s') Z(s', \psi(n), o) V^\pi(\eta(n, o), s') \tag{6}$$

$V^\pi(n, s)$ is the value of state s of the α-vector corresponding to the node n:

$$V^\pi(n, s) = \alpha_n(s) \tag{7}$$

The runtime of the policy evaluation step can be under $(|\mathcal{N} \times |S|)^3$.

Policy Improvement

In the Policy Improvement step, to improve a FSC π, Hansen's algorithm updates each α-vector from the current set of α-vectors $\mathcal{V}$, and then constructs an updated controller π' from these new α-vectors. To achieve this, first a DP- update as in algorithm (1) is performed on $\mathcal{V}$ to get the set $\mathcal{V}'$ corresponding to the next horizon. π' is then computed by incorporating these vectors in π: for each $\alpha' \in \mathcal{V}'$,

- If the action and successor links of α' are identical to that of some node originally in π, then the node remains unchanged in π
- If α' pointwise dominates some nodes in π, they are replaced by a node corresponding to α'

Algorithm 2: General Policy Iteration Algorithm

Function `Policy_Iteration(`π_0`)`

Input : Initial FSC π_0
Output: Optimal FSC π'

$\pi \leftarrow \pi_0$
while *True*
  //Policy evaluation step (equations 7 and 8) to compute α-vectors
  $(V) \leftarrow policyEvaluate(\pi)$
  $((N, \psi, \eta) \leftarrow \pi$
  $\pi' \leftarrow \pi$

  //Policy Improvement
  $\mathcal{V}' \leftarrow Value\_Update((V))$
  foreach $\alpha' \in (V)'$ **do**
    if $\exists n \in \mathcal{N} s.t. \alpha' \equiv n$ **then**
      continue
    else if $\exists n \in \mathcal{N} s.t. \alpha' >_p n$ **then**
      //Symbol $>_p$ indicates point-wise domination
      //add α' to π' as a node $\hat{n}$
      $\mathcal{N}_d \leftarrow \{n | n \in \mathcal{N}, \alpha' >_p n\}$
      delete all $n \in \mathcal{N}_d$ from π'
      foreach $n' \in (N)'$ **do**
        foreach $o \in O$ **do**
          if $\eta(n', o) \in \mathcal{N}'_d$ **then**
            $\eta(n', o) \leftarrow \hat{n}$
    else
      add α' to π' as a node $\hat{n}$
  //Pruning
  $\mathcal{N}_f \leftarrow (N)' - \mathcal{N}$
  foreach $n \in \mathcal{N}' - \mathcal{N}_f$ **do**
    if $\forall n_f \in \mathcal{N}_f, n$ *is not reachable from* n_f **then**
      delete n from π';
  if $\pi' = \pi$ **then**
    break
  else
    $\pi \leftarrow \pi'$
return π'

- Else, a node is added to π' that has the same action and observation strategy as that of α'.

Before the next policy evaluation step, any node in π' which has no corresponding α-vector in $\mathcal{V}'$ is pruned, as long as the node is not reachable from a node which has a associated vector in $\mathcal{V}'$. Since the algorithm chooses to use all possible updates in every iteration, we call them Howard-like updates: based on Ronald Howard's MDP policy iteration [6], which used a similar approach. In the worst case, the size of $\mathcal{V}'$ can be proportional to $|A||\mathcal{V}|^{|O|} = |A||\mathcal{N}|^{|O|}$. Thus, one of our objective is to modify the Policy Improvement step so that it only uses a subset of $\mathcal{V}'$

The optimal POMDP policy, if it exists is defined by the following optimality condition for its value function: [10, 5]

$$V^*(n,s) = \max_{a \in A} \left[\sum_{s' \in S} T(s,a,s') R(s,a,s') + \gamma \sum_{s' \in S} \sum_{o \in O} T(s,a,s') Z(s',a,o) V^*(\eta(n,o),s') \right] \tag{8}$$

4.2.1 Hansen's Policy Iteration

The policy improvement step of Hansen's policy iteration [4] involves dynamic programming to transform the value function V^π represented by Γ_π into an improved value function represented by another set of α-vectors, $\Gamma_{\pi'}$. To get Hansen's Policy Iteration algorithm, we can replace the Value_Update function in the general policy iteration algorithm 2 by the function that return the DP_Update of a set of α-vectors.

4.2.2 Point Based Policy Iteration (PBPI)

The policy improvement step of PBPI [7] replaces the DP_Update algorithm of the general policy iteration algorithm (2) by the backup function described in algorithm (1). This combines the desirable properties of Hansen's policy iteration with point based value iteration. The set of belief points $\hat{B}$ is initialised with $\hat{B} = \{b_0\}$. The set of belief points is then updated at every iteration to include more points that are reachable from the current set of belief points in a single step.

5 Subset Update

To reduce the size of π', one strategy could be to select only a random subset of $\mathcal{V}'$ be incorporated into π in the Policy Improvement step as suggested in the BTP report by Karkhanis and Kalyanakrishnan [8]. The subset update alogrithm from the [8] is described in algorithm (3).

Algorithm 3: Modified Policy Improvement

Function `Subset_Policy_Improvement` $(\pi, bel, B, \hat{N})$

Input : Initial controller π, Belief bel, Branching Factor B, Node Limit $\hat{N}$
Output: Improved controller π', Boolean var whether $|\mathcal{N}\ \hat{N}$

$maxReward \leftarrow -\infty$
$pi' \leftarrow \pi$
$(\mathcal{N}, \psi, \eta) \leftarrow \pi$
//Stores if node limit is very close $conv \leftarrow 0$
for $b \leftarrow 1$ **to** B **do**
  $conv_1 \leftarrow 0$
  $\pi_1 \leftarrow \pi$
  get α-vectors of π as $\mathcal{V}$
  $\mathcal{V}' \leftarrow Value\_Update(\mathcal{V})$
  $subV \leftarrow \phi$
  foreach $v \in (V)'$ **do**
    //Choose uniformly at random from {0, 1}
    $x = boolRand(0.5)$
    if $x = 1$ **then**
      $subV \leftarrow subV \cup \{v\}$
  //If no suitable subset was found till the last try, use all vectors
  if $b = B$ ***and*** $\pi' = pi$ **then**
    $subV \leftarrow \mathcal{V}'$

  //a smaller set, $subV$ added instead of $\mathcal{V}'$
  foreach $\alpha_1 \in subV$ **do**
    //α_1 picked randomly each time without replacement
    if $\alpha_1 \in \mathcal{V}$ **then**
      continue
    else
      if *size of* $\pi_1 \leq \hat{N}$ **then**
        add node representing α_1 to π_1
      else
        $conv_1 \leftarrow 1$
        break
  foreach $n, n' \in \pi_1$ *s.t.* $\alpha_n \geq_p \alpha_{n'}$ **do**
    redirect incoming edges of n' in π_1 to n
    delete n' from π_1

  $\mathcal{V}_1 \leftarrow policyEvaluate(\pi_1)$
  $currReward \leftarrow max_{\alpha \in \mathcal{V}_1}\ [bel \cdot \alpha]$
  if $currReward > maxReward$ **then**
    $maxReward \leftarrow currReward$
    $\pi; \leftarrow \pi_1; conv \leftarrow conv_1$
return $\pi', conv$

6 FSC Pruning

Since the initial belief state b_0 is known during the planning, we can use this information to prune the FSC in each iteration. Say, the node n_0 gives the maximum expected reward for this belief. That is,

$$n_0 = \underset{n \in \mathcal{N}}{\operatorname{argmax}} \left[b_0 \cdot \alpha_n \right] \tag{9}$$

Then we can delete all nodes n from π which are not reachable from n_0, without affecting the expected reward for b_0. This can be done before every iteration.

7 Union FSC

A potentially effective approach to reach better policies would be by combining two different FSCs into one such that the resultant FSC is able to use the properties learned in both the policies. If this is done efficiently, multiple FSCs could be parallelly computed, each focussed on different improvements, and later combined. A trivial way to do this would be by assuming the set of nodes of the combined FSC to be a union of the set of nodes of either FSCs. This leads to a larger combined FSC $\hat{\pi}$ which has two disconnected components, each corresponding to one of the original FSCs. Although such a formulation does give a usable FSC, it is not desirable for infinite horizon problems. This is because the reason Policy Iteration works well (converges in fewer steps than value iteration) is that it takes newly learned strategies (eg. one step lookahead in policy improvement) which are known to be better if used once and modifies the policy such that they are used every time (ensured by pruning dominated states) the same situation (belief) arises. To counteract this problem, we can keep a track of the belief state while using the union of the FSCs. Therefore, at each step:

- Take the action corresponding to the node which promises the highest minimum expected reward from either of the FSCs. The minimum expected reward will be the dot product of the current belief state with the α-vector corresponding to the node.
- Now, based on the observation received, update the current belief state and repeat.

The reason the dot product with any α_n is the minimum expected reward and not the actual expected reward is because the subsequent steps might lead to choosing a node which is different (but better) than the successor $\eta(n, o)$. Thus, the actual expected reward might be greater.

8 Experiments

8.1 POMDP Problems

8.1.1 Problem 1: Marketing Decisions

A company needs to decide at each time-step if either to market a Luxury product(L) or a standard product(S). The action will affect the brand preference (B) of the consumers. However, the company can only observe whether the product is purchased (P) or not. The equivalent POMDP representation has $|S| = |O| = |A| = 2, \gamma = 0.95, b_0 = (0.5)$. Refer to [3] for further details.

Actions	Transition Probabilities	Observation Probabilities	Expected Reward
Market Luxury Product (L)			

	B	∼B
B	0.8	0.2
∼B	0.5	0.5

	P	∼P
B	0.8	0.2
∼B	0.6	0.4

B	∼B
4	-4

Market Standard Product (S)

	B	∼B
B	0.5	0.5
∼B	0.4	0.6

	P	∼P
B	0.9	0.1
∼B	0.4	0.6

B	∼B
0	-3

8.1.2 Problem 2: 4x3 Grid Navigation

The objective is to navigate a robot to a goal (G) through a 4x3 maze. Falling into the trap (T) incurs a penalty. 0s indicate free spaces, 1s indicate obstructions/walls. It is equally likely to start anywhere The actions, NSEW, have the expected result 80% of the time, and 20% of the times cause a transition in a direction perpendicular to the intended (10% for each direction).

Movement into a wall returns the robot to its original state. Robot's observations are limited to two wall detectors that can detect when a wall is to the left or right. This gives 6 possible observations. Refer to [5] for further details.

The Maze	Rewards	States	Actions	Observations	Discount

0	0	0	G
0	1	0	T
0	0	0	0

G	T
+1	-1

$|\mathcal{S}|$=11 Obstacle not counted

NSEW

left, right, neither, both, good, bad, and absorb

$\gamma = 0.95$

8.1.3 Problem 3: Rock Sample

The RockSample problem introduced in [11] models a rover that can achieve reward by sampling rocks in the immediate area, and by continuing its traverse (reaching the exit at the right side of the map). The positions of the rover and the rocks are known, but only some of the rocks have scientific value; such rocks are called "good". Sampling a rock is expensive, so the rover is equipped with a noisy long-range sensor that it can use to help determine whether a rock is good before choosing whether to approach and sample it.

An instance of RockSample with map size $n \times n$ and k rocks is described as RockSample[n, k]. The POMDP model of RockSample[n, k] is as follows. The state space is the cross product of $k + 1$ features: Position = {(1, 1), (1, 2), ... , (n, n)}, and k binary features $RockType_i = \{Good, Bad\}$ that indicate which of the rocks are good. There is an additional terminal state, reached when the rover moves off the right-hand edge of the map. The rover can select from $k+5$ actions: $\{North, South, East, West, Sample, Check_1, ..., Check_k\}$. The first four are deterministic single-step motion actions. The Sample action samples the rock at the rover's current location. If the rock is good, the rover receives a reward of 10 and the rock becomes bad (indicating that nothing more can be gained by sampling it). If the rock is bad, it receives a penalty of -10. Moving into the exit area yields reward 10. All other moves have no cost or reward.

Each $Check_i$ action applies the rover's long-range sensor to rock i, returning a noisy observation from $\{Good, Bad\}$. The noise in the long-range sensor reading is determined by the efficiency η, which decreases exponentially as a function of Euclidean distance from the target. At $\eta = 1$, the sensor always returns the correct value. At $\eta = 0$, it has a 50/50 chance of returning Good or Bad. At intermediate values, these behaviors are combined linearly. The initial belief is that every rock has equal probability of being Good or Bad.

Grid	Rewards	States	Acions	Observations	Discount
$n \times n$ Grid	Sample Good Rock: +10 Sample Bad Rock: -10	$\|S\| = n^2 \cdot 2^k + 1$	N,S,E,W, Sample, $Check_1$, ..., $Check_k$	Good,Bad, None	$\gamma = 0.95$

8.2 Libraries Used

8.2.1 AI-Toolbox

The C++ library AI-Toolbox [1] has been used to write the code for the POMDP planning algorithms. The code for PBPI with subset update is written using this library. The library simulates the planning algorithm

given a POMDP problem instance as the 8-tuple $(S, A, T, O, Z, \kappa, b_0, \gamma)$.

8.2.2 Eigen

The C++ library Eigen [2] has been used to solve the Policy Evaluation step. The system of linear equations for the policy evaluation step was encoded as a sparse matrix. The system of linear equations were then solved using the QR decomposition of the sparse matrix which is significantly faster than solving the system of equation with any other non-sparse method.

8.2.3 POMDP_PY

The python library POMDP_PY [13] has been used to generate the instances of the rocksample problem. Since the library doesn't provide a POMDP problem instance as an 8-tuple but instead as python functions, the output generated by the library was converted to 8-tuple instance of POMDP problems by enumerating over all possible states, actions and observations to obtain the transition matrix and observation matrix.

8.3 Results

The following results were obtained upon using Subset Update (Algorithm 3) and Union FSC with PBPI. The fourth column contains the results obtained by running PBPI with subset update and taking the union of 5 different FSCs. The input Node Limit $\hat{N}$ = 100 for all the problems. The initial FSC π_0 was the one state self loop FSC with α-vector = 0. Initial belief bel is a uniform probability distribution over all states except the goal and trap states for the 4x3 maze problem (probability of being in the goal or trap is 0 initially). The table given below shows the average expected reward achieved by each of the alogrithmic variants. The results were obtained by simulating the policy obtained over a horizon of 500 timesteps. Further, the simulations were run over 5 different random seeds and the average and maximum rewards acheived over the 5 different random seeds have been reported. Conv. represents that the particular algorithm converged to the maximum expected reward for the problem.

Problem Number	PBPI		PBPI w/ Subset Update		PBPI w/ Subset Update(Union FSC)
	avg	max	avg	max	
Problem 1	Conv.	Conv.	Conv.	Conv.	Conv.
Problem 2	2.168	2.247	2.185	2.452	2.427
Problem 3	16.942	16.942	17.062	17.062	17.621

From the results, we can observe that PBPI with subset update gives a marginal increase in the expected reward over PBPI without subset update. Taking the union of 5 different FSCs gives better expected rewards

on average. However, even though the union FSC seems to give better results than PBPI with subset update, the increase in their expected reward (avg) is misleading. This is because the union FSC should ideally be compared with the maximum expected reward of the FSC rather than the average expected rewards of different FSCs. When comparing union FSC with the max value for PBPI with subset update, we see that the union FSC does not necessarily improve the expected reward over PBPI with subset update. The expected reward for union FSC stays in the range of the maximum expected reward of the individual FSCs it is made up of.

9 Conclusion

This text takes forward the research done by Karkhanis and Kalyanakrishnan [8]. As shown in their text, the subset update modification in the policy improvement step works well for Hansen's policy iteration. However, similar results were not obtained for the subset update modification in PBPI. One of the reasons could be that since Hansen's policy iteration is a deterministic algorithm, the subset update modification introduces randomization in it that helps in bringing more variability to the FSC while not letting the size of the FSC to increase drastically. PBPI on the other hand already has randomization in it due to the belief points set. PBPI also has a control over the growth of FSC size by limiting the size of the belief points set. Hence, it could be possible that the subset update algorithm for PBPI doesn't improve the expected reward by much due to the above reasons. Moreover, the union FSC algorithm also doesn't give better results when used with FSCs obtained from PBPI. The policy obtained by the union FSC is very similar to the policy of the FSC with the highest expected reward. This behaviour suggests that the variability introduced by changing the random seed is not strong enough for the union FSC to leverage all the individual FSCs in the union.

The approximation introduced by PBPI in the policy improvement step lets it scale to POMDPs with a large number of states. However, the algorithm still takes a large amount of time to approach to a good enough policy for POMDP with an even larger number of states. There is a still a need for an algorithm that could scale well with the size of POMDP and could also be parallelized across multiple computing resources. The union FSC not giving good results with PBPI suggests that we need to look for newer ways to tackle this problem.

References

[1] Eugenio Bargiacchi, Diederik M. Roijers, and Ann Nowé. "AI-Toolbox: A C++ library for Reinforcement Learning and Planning (with Python Bindings)". In: *Journal of Machine Learning Research* 21.102 (2020), pp. 1–12. URL: http://jmlr.org/papers/v21/18-402.html.

[2] Gaël Guennebaud, Benoît Jacob, et al. *Eigen v3*. http://eigen.tuxfamily.org. 2010.

[3] Eric Hansen. "An Improved Policy Iteration Algorithm for Partially Observable MDPs". In: *Advances in Neural Information Processing Systems*. Ed. by M. Jordan, M. Kearns, and S. Solla. Vol. 10. MIT Press, 1998.

[4] Eric A. Hansen. "An Improved Policy Iteration Algorithm for Partially Observable MDPs". In: *Conference on Neural Information Processing Systems* (1993), pp. 1015–1021.

[5] Eric Anton Hansen and Shlomo Zilberstein. "Finite-Memory Control of Partially Observable Systems". AAI9909170. PhD thesis. 1998. ISBN: 0599073322.

[6] R. A. Howard. *Dynamic Programming and Markov Processes*. Cambridge, MA: MIT Press, 1960.

[7] Shihao Ji et al. "Point-Based Policy Iteration". In: *Proceedings of the 22nd National Conference on Artificial Intelligence - Volume 2*. AAAI'07. Vancouver, British Columbia, Canada: AAAI Press, 2007, pp. 1243–1249. ISBN: 9781577353232.

[8] Deep Karkhanis and Shivaram Kalyanakrishnan. *Tractable Policy Iteration in POMDPs*. 2020.

[9] Joelle Pineau, Geoffrey J. Gordon, and Sebastian Thrun. "Anytime Point-Based Approximations for Large POMDPs". In: *CoRR* abs/1110.0027 (2011). arXiv: 1110.0027. URL: http://arxiv.org/abs/1110.0027.

[10] Richard D. Smallwood and Edward J. Sondik. "The Optimal Control of Partially Observable Markov Processes Over a Finite Horizon". In: *Operations Research* 21.5 (1973), pp. 1071–1088. ISSN: 0030364X, 15265463. URL: http://www.jstor.org/stable/168926.

[11] Trey Smith and Reid G. Simmons. "Heuristic Search Value Iteration for POMDPs". In: *CoRR* (2012). arXiv: 1207.4166. URL: http://arxiv.org/abs/1207.4166.

[12] Matthijs T. J. Spaan and Nikos A. Vlassis. "Perseus: Randomized Point-based Value Iteration for POMDPs". In: *CoRR* abs/1109.2145 (2011). arXiv: 1109.2145. URL: http://arxiv.org/abs/1109.2145.

[13] Kaiyu Zheng and Stefanie Tellex. "pomdp_py: A Framework to Build and Solve POMDP Problems". In: *ICAPS 2020 Workshop on Planning and Robotics (PlanRob)*. Arxiv link: "https://arxiv.org/pdf/2004.10099.pdf". 2020. URL: https://icaps20subpages.icaps-conference.org/wp-content/uploads/2020/10/14-PlanRob_2020_paper_3.pdf.