Policy Iteration in POMDPs

Deep Karkhanis
deep_karkhanis@iitb.ac.in

Shivaram Kalyanakrishnan
shivaram@cse.iitb.ac.in

B.Tech Project

IIT-Bombay
December 2019

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 proposes a variant of Policy Iteration in POMDPs, which makes this solving more tractable. We specify a method which regulates the update of Finite-State Controllers (FSCs) in Hansen's Policy Iteration algorithm [1]. We selectively add only a subset of improving FSC nodes (as opposed to adding all improvements), during Policy Improvement. Towards the end of the text, we also suggest a method to locally combine FSC nodes, in order to decrease controller size without impairing the policy.
The 2nd and 3rd section in the text formally define MDPs and POMDPs and also mention how POMDP solutions (formally called policies) are represented. Important existing POMDP solving algorithms are summarized in section 4. The initial insights gained during the project are mentioned in section 5. Section 6 describes a set of theorems which are necessary for the correctness of the algorithm called Subset Update that this text proposes in section 7. Section 9 summarizes the results of experiments conducted as part of the project. Sections 8 and 10 describe variations and future prospects of work possible for the project.

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 commonly used formal model to represent uncertain but fully-observable environments. An MDP is a 6-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{K}, s_0, \gamma)$. Where,

- $\mathcal{S}$ is a finite set of states with $s_0 \in \mathcal{S}$ as the agent's initial state
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the transition function such that $\forall s \in \mathcal{S}, a \in \mathcal{A} : \mathcal{T}(s, a, .)$ is a probability distribution over $\mathcal{S}$
- $\mathcal{K} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ 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 $\mathcal{A}$, which makes it randomly change its state from $s_t \in \mathcal{S}$ to $s_{t+1} \in \mathcal{S}$, according to the pdf $\mathcal{T}(s_t, a_t, .)$. 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 $\mathcal{T}$, 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 $\mathcal{Z} : \mathcal{S} \times \mathcal{A} \times \mathcal{O} \to \mathbb{R}$, such that $\forall s \in \mathcal{S}, a \in \mathcal{A} : \mathcal{Z}(s, a, .)$ is a probability distribution over $\mathcal{O}$. $\mathcal{Z}(s, a, o)$ 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 $\mathcal{Z}(s_{t+1}, a_t, .)$. The goal remains the same–maximize the infinite horizon expected reward:

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

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 $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{Z}, \mathcal{K}, b_0, \gamma)$. The belief state is also represented as a vector of size $|\mathcal{S}| - 1$ representing the probabilities of the first $|\mathcal{S}| - 1$ states. The belief space $\mathcal{B} = [0, 1]^{|\mathcal{S}|-1}$ is a real-valued set in the $(|\mathcal{S}| - 1)$-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).

3.1 α-vectors and Finite State Controllers

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.
- $\psi : \mathcal{N} \to \mathcal{A}$ is the action selection function that for each node n prescribes an action $\psi(n)$
- $\eta : \mathcal{N} \times \mathcal{O} \to \mathcal{N}$ 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 $|\mathcal{S}|$, 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 $\mathcal{R}(b)$ of:

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

Here, function $\mathcal{R} : \mathcal{B} \to \mathbb{R}$ maps the belief states to their expected reward according to the FSC. $\mathcal{R}$ 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

$V^\pi : \mathcal{N} \times \mathcal{S} \to \mathbb{R}$, 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.

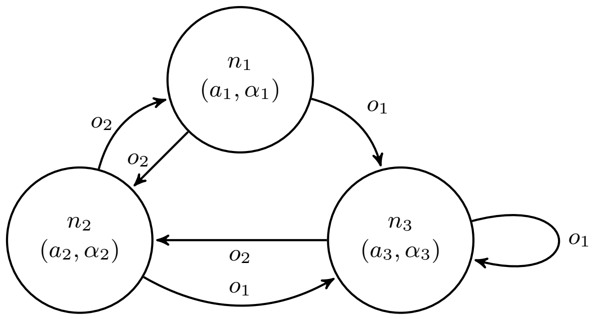

Figure 1: A sample FSC with $\mathcal{N} = \{n_1, n_2, n_3\}$. $\psi(n_i) = a_i$, $\forall i \in \{1, 2, 3\}$. Each node's successors for observations in $\mathcal{O} = \{o_1, o_2\}$ are denoted by edges. α_i is the α-vector corresponding to node n_i $\forall i \in \{1, 2, 3\}$.

4.1 Value Iteration

Value iteration is a method in which one keeps updating a set of α-vectors, by what is known as a Dynamic Programming Update (DP-update), until the set cannot be updated further (indicating convergence).

One starts with a initial set of α-vectors $\mathcal{V}_0$. Generally, $\mathcal{V}_0 = \{\vec{0}\}$ (i.e a singleton set with a zero-vector). In any iteration t, the algorithm looks one horizon further, and performs a DP-update of the set $\mathcal{V}_t$ to $\mathcal{V}_{t+1}$:

Algorithm 1: Dynamic Programming Update

```
DP_update ($\mathcal{V}_{in}$)
    Input: Set of $\alpha$-vectors $\mathcal{V}_{in}$
    Result: $\alpha$-vectors for the next horizon $\mathcal{V}_{out}$

    $\mathcal{V}_{out} \leftarrow \phi$;
    forall $a \in \mathcal{A}$ do
        forall $nxt\_str \in \mathcal{V}_{in}^{|\mathcal{O}|}$ do
            new_$\alpha \leftarrow \vec{0}$;
            forall $s \in \mathcal{S}$ do
                /* nxt_str(o')[a] is the successor $\alpha$-vector for o'
                   nxt_str(o')(s') is the value of that vector for state s' */

                new_$\alpha(s) \leftarrow \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') \left[\mathcal{K}(s, a, s') + \gamma \sum_{o' \in \mathcal{O}} \mathcal{Z}(s', a, o') nxt\_str(o')(s')\right]$
            end
            $\mathcal{V}_{out} \leftarrow \mathcal{V}_{out} \cup \{new\_\alpha\}$;
        end
    end
    //Pruning $\mathcal{V}_{out}$ to a Parsimonious Set
    forall $\alpha \in \mathcal{V}_{out}$ do
        remove $\alpha$ from $\mathcal{V}_{out}$ if not useful;
    end
    return $\mathcal{V}_{out}$;
```

[a]Note that nxt_str is a $|\mathcal{O}|$-tuple of successor α-vectors. To avoid complicated notations, we have assumed that the successor vector for each observation can directly be referenced as nxt_str(o). Ideally, one would use a indexing function which maps each $o \in \mathcal{O}$ to a unique value between 1 to $|\mathcal{O}|$ and use this index to get the α-vector.

In order to calculate $\mathcal{V}_{t+1}$, the method described above enumerates all possible actions and observation mappings to the set $\mathcal{V}_t$.

White and Harrington [3] showed that the finite horizon expected reward function $\mathcal{R}$ corresponding to each such set of vectors is piece-wise linear and convex. Not all vectors in the set $\mathcal{V}_{t+1}$ are useful. A vector will only be useful if for some belief state, the vector promises a expected reward which is higher than the reward promised by any other vector. Thus, it is a common practice to prune the set $\mathcal{V}_{t+1}$ to a parsimonious set $\mathcal{V}'_{t+1}$ in each iteration. A set of α-vectors $\mathcal{V}'$ is parsimonious if:

$$\forall \alpha \in \mathcal{V}',\ \exists b \in \mathcal{B}\ s.t. \quad b \bullet \alpha > b \bullet \alpha' \quad \forall \alpha' \in (\mathcal{V}' - \{\alpha\}) \tag{5}$$

Various algorithms differ in the way they calculate the set ${\mathcal{V}'}_{t+1}$. Some algorithms calculate ${\mathcal{V}'}_{t+1}$ by generating $\mathcal{V}_{t+1}$ of size $|\mathcal{A}||\mathcal{V}|^{|\mathcal{O}|}$ and then pruning dominated α-vectors, usually by linear programming. Such algorithms include Monahan's algorithm [4], and Incremental pruning [5]. Monahan's algorithm looks at all possible combinations (pairs) to find dominated vectors. Incremental Pruning first prunes vectors by comparing those which have the same action. To do this, it starts combining vectors by incrementally looking at minimal difference in observations. When combining across actions, it keeps a combined set initialized with the first action set. In each step, it merges any one of the remaining action set with the combined set.

Other methods build the set ${\mathcal{V}'}_{t+1}$ directly from the previous set $\mathcal{V}_t$, by only considering useful conditional plans. Sondik's One-pass algorithm [6] and [7] generates a new vector by choosing a belief state which has not yet been assigned its optimal vector and computes the same for it. It then finds the region around this belief where the new vector will surely dominate and adds the vector to the final set. It thus expands the said set until all belief regions have been covered. Cheng's Linear Support Algorithm [8] generates new vectors by looking at the edges of the belief space partitions formed due to the existing set of vectors (regions where a particular vector dominates all other vectors in the current set). If none of the edge points give a new vector, the set has converged. Witness Algorithm [9] generates separate sets for each action and combines them. While building a set for a particular action, it searches for "witness" regions where a modification of the current vector's strategy could provide a better reward.

Note that for some POMDPs, Value Iteration algorithms might need infinite number of iterations to converge.

4.2 Policy Iteration

Unless a value iteration algorithm described above converges for a given POMDP, the set of vectors 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 [1] and [2], is an iterative algorithm which repeatedly performs two steps: Policy Evaluation and Policy Improvement, until convergence. The algorithm 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' is a strict improvement of FSC π iff:

$$(\forall b \in \mathcal{B} \quad \mathcal{R}'(b) \geq \mathcal{R}(b)) \wedge (\exists b' \in \mathcal{B} \quad \mathcal{R}'(b') > \mathcal{R}(b')) \tag{6}$$

where $\mathcal{R}$ and $\mathcal{R}'$ 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.

Algorithm 2: Hansen's Policy Iteration Algorithm

```
Policy_Iteration ($\pi_0$)
    Input: Initial FSC $\pi_0$
    Result: Optimal FSC $\pi'$

    $\pi \leftarrow \pi_0$;
    while True do
        //Policy evaluation step (equations 7 and 8) to compute $\alpha$-vectors
        $\mathcal{V} \leftarrow policyEvaluate(\pi)$;
        $(\mathcal{N}, \psi, \eta) \leftarrow \pi$;
        $\pi' \leftarrow \pi$;

        //Policy Improvement
        $\mathcal{V}' \leftarrow DP - update(\mathcal{V})$;
        forall $\alpha' \in \mathcal{V}'$ 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 $\alpha'$ to $\pi'$ as a node $\widehat{n}$;
                $\mathcal{N}_d \leftarrow \{n | n \in \mathcal{N}, \alpha' >_p n\}$;
                delete all $n \in \mathcal{N}_d$ from $\pi'$;
                forall $n' \in \mathcal{N}'$ do
                    forall $o \in \mathcal{O}$ do
                        if $\eta(n', o) \in \mathcal{N}_d$ then
                            $\eta(n', o) \leftarrow \widehat{n}$;
                    end
                end
            else
                add $\alpha'$ to $\pi'$ as a node $\widehat{n}$;
        end
        //Pruning
        $\mathcal{N}_f \leftarrow \mathcal{N}' - \mathcal{N}$;
        forall $n \in \mathcal{N}' - \mathcal{N}_f$ do
            if $\forall n_f \in \mathcal{N}_f$, n is not reachable from $n_f$ then
                delete $n$ from $\pi'$;
        end
        if $\pi' = \pi$ then
            break;[a]
        else
            $\pi \leftarrow \pi'$;
    end
    return $\pi'$;
```

[a] See [1] for other terminating conditions

Policy Evaluation

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

$$V^\pi(n,s) = \sum_{s' \in \mathcal{S}} T(s,\psi(n),s')R(s,\psi(n),s') + \gamma \sum_{s' \in \mathcal{S}} \sum_{o \in \mathcal{O}} T(s,\psi(n),s')Z(s',\psi(n),o)V^\pi(\eta(n,o),s') \quad (7)$$

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

$$V^\pi(n,s) = \alpha_n(s) \quad (8)$$

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

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 (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'
- 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 [10], which used a similar approach.

In the worst case, the size of $\mathcal{V}'$ can be proportional to $|\mathcal{A}||\mathcal{V}|^{|\mathcal{O}|} = |\mathcal{A}||\mathcal{N}|^{|\mathcal{O}|}$. Thus, our objective is to modify the Policy Improvement step so that it only uses a subset of $\mathcal{V}'$

4.3 Partially Observable Monte Carlo Planning

David and Joel [11] introduced a Monte-Carlo algorithm for online planning in POMDPs, called the Partially Observable Monte Carlo Planning algorithm or POMCP. The algorithm combines a Monte-Carlo update of the agent's belief state with a Monte-Carlo tree search (MCTS) starting from the current belief state. The rollout strategy during MCTS involves using a POMDP simulator. This POMCP algorithm, has two important properties. First, Monte Carlo sampling works around the constraints of dimensionality, both during belief state updates and during planning. Second, only a black box simulator of the POMDP is required, rather than explicit probability distributions. These properties enable POMCP to plan effectively in a larger class of POMDPs.

4.4 Recent Context

Various POMDP solving algorithm exist which build on the ones discussed above. The survey [12] provides a more detailed analysis. Currently, the most pressing issue in POMDP solvers has been that of scalability. Although offline solvers can make use of recent advances in computational powers, most of them can only be used for solving toy POMDP problems. The Policy Iteration algorithm proposed by this text tries to address this scalability issue by permitting the solver to be in control of the blowup in each iteration. Although this leads to smaller policy improvements in each iteration, the computational advantages allow for much better policies to be learned compared to Hansen's (Howard-like) Policy Iteration [1].

5 Initial Analysis

After implementing Hansen's version of Policy Iteration, some analysis was done to find general trends in rewards. Based on multiple executions and iterations of the algorithm, the following points were found to be empirically true:

- The FSC obtained after each iteration dominates the older FSC, unless the algorithm has converged
- A larger FSC size correlates with a better controller

The results in Figure (2 and 3) were obtained for the 4x3 Grid Problem (9.1.2)

6 Proof of Policy Improvement

Theorem 1. *If a controller π' can be obtained simply by addition of nodes to a controller π, and the expected rewards for any Belief State b of the POMDP, are $\mathcal{R}'(b)$ for π' and $\mathcal{R}(b)$ for π, then:*

$$\mathcal{R}'(b) \geq \mathcal{R}(b), \quad \forall\, b \in \mathcal{B} \tag{9}$$

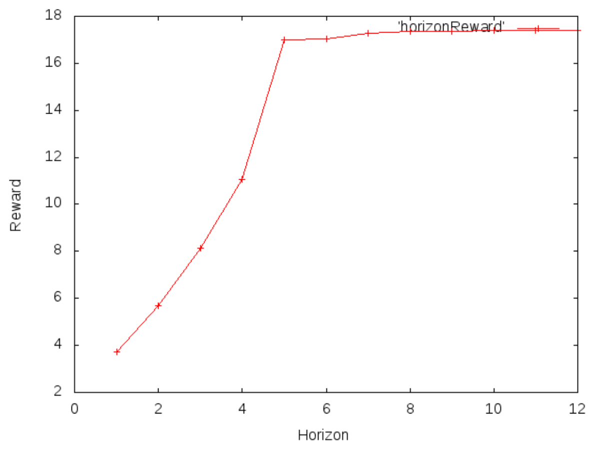

Figure 2: Increasing Reward in each iteration

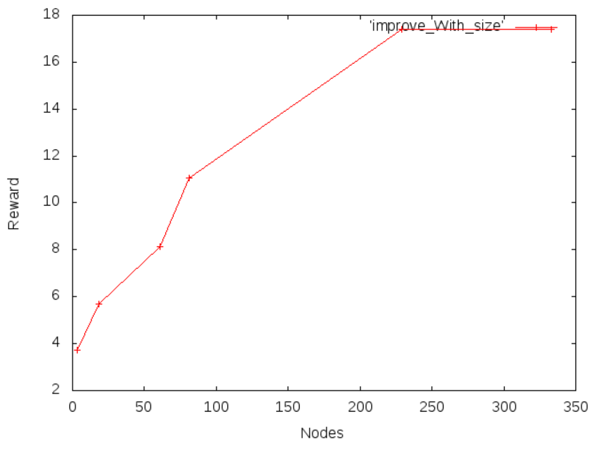

Figure 3: Larger FSC-size correlates with a better policy

Remark 1. *To define a node n completely, both its action $\psi(n)$ and successor nodes $\eta(n, o)$ for every observation o need to be specified. Thus, the set of nodes $\mathcal{N}$ of the FSCs $\pi = (\mathcal{N}, \psi, \eta)$ and $\pi' = (\mathcal{N}', \psi', \eta')$ follow:*

$$\begin{aligned}
&\mathcal{N} \subseteq \mathcal{N}' \\
&\psi(n) = \psi'(n), \forall n \in \mathcal{N} \\
&\eta(n, o) = \eta'(n, o), \forall n \in \mathcal{N}, \forall o \in \mathcal{O}
\end{aligned} \tag{10}$$

Proof.
Let the set of Linear Equations in Policy Evaluation (equation 7) for π' are $\mathcal{E}'$ and for π are $\mathcal{E}$. From the equations in Remark 1, it is clear that $\mathcal{E} \subseteq \mathcal{E}'$. Thus,

$$V^{\pi'}(n, s) = V^{\pi}(n, s) \;\; \forall n \in \mathcal{N}, s \in \mathcal{S} \tag{11}$$

Thus, using (4):

$$\alpha'_n = \alpha_n \;\; \forall n \in \mathcal{N} \tag{12}$$

Since $\mathcal{N} \subseteq \mathcal{N}'$ and using equation (12), $\forall\; b \in \mathcal{B}$:

$$\max_{n \in \mathcal{N}'} [b \bullet \alpha'_n] \;\geq\; \max_{n \in \mathcal{N}} [b \bullet \alpha'_n] \;\geq\; \max_{n \in \mathcal{N}} [b \bullet \alpha_n]$$

Thus, using equation (3):

$$\mathcal{R}'(b) \geq \mathcal{R}(b)$$

Note, an important assumption taken in the proof is that the policy evaluation equations (7) for nodes $n \in (\mathcal{N}' - \mathcal{N})$ have a solution. This will be true if $\mathcal{V}'$ is obtained from a DP update of $\mathcal{V}$. This is because for such an n: $\eta(n, o) \in \mathcal{N}$, $\forall\; o \in \mathcal{O}$. That is, the terms on the RHS of the equations are already known from $\mathcal{V}$.

Theorem 2. *If $\mathcal{V}'$ is the parsimonious set of α-vectors obtained from the DP-update (algorithm 1) of a sub-optimal FSC $\pi = (\mathcal{N}, \psi, \eta)$, and the FSC $\pi' = (\mathcal{N}', \psi', \eta')$ is obtained by simply adding to π, the vectors in set $\mathcal{V}'_1$ such that $\mathcal{V}'_1 \subseteq \mathcal{V}'$ and π' has atleast one node different from π, then–*

$$\begin{aligned}
\exists\; \alpha' \in \mathcal{V}'_1,\; b \in \mathcal{B} \; s.t. \quad & [b \bullet \alpha'] > \max_{n \in \mathcal{N}} [b \bullet \alpha_n] \\
thus, \quad & \mathcal{R}'(b) > \mathcal{R}(b)
\end{aligned} \tag{13}$$

Note: $\mathcal{R}$ and $\mathcal{R}'$ are the reward functions as in Lemma (1) and $\mathcal{V}$ is the set of α-vectors corresponding to $\mathcal{N}$

Proof.

Since π' has atleast one new node: $\exists\ n' \in \mathcal{N}'$ such that $n' \notin \mathcal{N}$. Say $\alpha_{n'}$ corresponds to the α-vector of n'.

Say $\widehat{\mathcal{V}}'$ is the set obtained in the DP-update, before pruning to a parsimonious set. Then, $\widehat{\mathcal{V}}'$ is the set of α-vectors which represents all possible first-step strategies available to the agent, given that it follows the FSC π second-step onwards. That is, $\widehat{\mathcal{V}}'$ exactly and exhaustively includes all possible α-vectors which have their action in $\mathcal{A}$ and their successors for every observation are present in $\mathcal{V}$. Since the nodes in π themselves represent valid strategies:

$$\mathcal{V} \subseteq \widehat{\mathcal{V}}' \tag{14}$$

Since $\mathcal{V}'$ is a parsimonious set,

$$\exists b' \in \mathcal{B}\ s.t.\ \forall \alpha \in (\mathcal{V}' - \{\alpha_{n'}\}) \quad [b' \bullet \alpha_{n'}] > [b' \bullet \alpha] \tag{15}$$

Moreover, since $\mathcal{V}'$ has been obtained by pruning non-useful nodes in $\widehat{\mathcal{V}}'$, $\mathcal{V}'$ and $\widehat{\mathcal{V}}'$ represent the same policy with $\mathcal{V}' \subseteq \widehat{\mathcal{V}}'$, and:

$$\forall \alpha \in (\widehat{\mathcal{V}}' - \{\alpha_{n'}\}) \quad [b' \bullet \alpha_{n'}] \geq [b' \bullet \alpha] \tag{16}$$

According to expression (15), for some $b' \in \mathcal{B}$, only $\alpha_{n'}$ gives the best expected reward, given that the agent follows FSC π second-step onwards. Moreover, there must be a continuous region $\mathcal{D}_{n'}$ in the belief space around b′, where $\alpha_{n'}$ must be the only vector giving the best reward i.e dominating. Since otherwise, b' would be on the boundary of a region dominated by $\alpha_{n'}$ and there would be another vector in $\mathcal{V}'$ (the vector sharing this boundary), which gives the same reward as $\alpha_{n'}$ at b'. This would contradict expression (15).

Now, if for any $\bar{\alpha} \in \widehat{\mathcal{V}}'$, the equality in expression (16) holds, then either $\exists b'' \neq b'$ in this continuous region $\mathcal{D}_{n'}$ where $\alpha_{n'}$ dominates $\bar{\alpha}$, or:

$$\alpha_{n'}(s) = \bar{\alpha}(s)\ \forall s \in \mathcal{S} \tag{17}$$

This is because if $\bar{\alpha}$ matches the expected reward of $\alpha_{n'}$ only for a finite number of $b \in \mathcal{D}_{n'}$, then any other point in this uncountable set is a suitable b''. Moreover, the only way the reward can match for uncountable number of $b \in \mathcal{D}_{n'}$, is if the values of the α-vectors are same. Thus, as long as the value of α-vectors do not match, expression (16) will be a strict inequality.

Since $n' \notin \mathcal{N}$ i.e it is a new node, $\forall \alpha \in \mathcal{V}$, $\alpha_{n'} \neq \alpha$. Thus, the expression (16) would be a strict inequality for vectors in $\mathcal{V}$. That is,

$$\begin{aligned} \forall \alpha \in \mathcal{V} \quad & [b' \bullet \alpha_{n'}] > [b' \bullet \alpha] \\ i.e. \quad \forall n \in \mathcal{N} \quad & [b' \bullet \alpha_{n'}] > [b' \bullet \alpha_n] \end{aligned}$$

Thus,

$$\max_{n \in \mathcal{N}'} [b' \bullet \alpha_n] > \max_{n \in \mathcal{N}} [b' \bullet \alpha_n]$$

Hence,

$$\mathcal{R}'(b') > \mathcal{R}(b')$$

Theorems 1 and 2 together show that as long as we only add vectors to π and atleast one vector is such that it was in $\mathcal{V}'$ but not originally in π, then the property of Policy Improvement is guaranteed. That is, the expected reward does not decrease for any belief state and increases for atleast one belief state.

Theorem 3. *If a n' and n are two nodes in a FSC $\pi = (\mathcal{N}, \psi, \eta)$ such that:*

$$\alpha_{n'} \geq_p \alpha_n \tag{18}$$

where, $\geq_p$ indicates pointwise domination or equality. Then, π', the FSC resulting upon deleting node n from π follows $\mathcal{R}^{\pi'} \geq_p \mathcal{R}^{\pi}$, that is:

$$\forall b \in \mathcal{B}, \quad \mathcal{R}^{\pi'}(b) \geq \mathcal{R}^{\pi}(b) \tag{19}$$

Proof.
The proof is based on the proof of Policy Iteration for MDPs [13] and [14].
Let $\boldsymbol{B} : (\mathcal{B} \to \mathbb{R}) \to (\mathcal{B} \to \mathbb{R})$ be the equivalent of the Bellman operator [15] for POMDPs. For the FSC π, it is $\boldsymbol{B}^{\pi}$ s.t:

$$\boldsymbol{B}^{\pi}(\mathcal{X})(b) = \sum_{s' \in \mathcal{S}} \mathcal{T}(s, \boldsymbol{ac}^{\pi}(b), s') \left[\mathcal{K}(s, \boldsymbol{ac}^{\pi}(b), s') + \gamma \sum_{o' \in \mathcal{O}} \mathcal{Z}(s', \boldsymbol{ac}^{\pi}(b), o') \mathcal{X}(b_{\boldsymbol{ac}^{\pi}(b), o'}) \right] \tag{20}$$

where, $\boldsymbol{ac}^{\pi}(b) = \psi(n_o)$ s.t. $\alpha_{n_o} = \arg\max_{\alpha \in \mathcal{V}^{\pi}} [b \bullet \alpha]$. i.e, $\boldsymbol{ac}^{\pi}$ specifies the action to be taken according to π for each belief b. Also, $b_{\boldsymbol{ac}^{\pi}(b), o'}$ is the belief state to which b will be updated upon performing $\boldsymbol{ac}^{\pi}(b)$ and observing o'

Lemma 1. *If $\mathcal{X} : \mathcal{B} \to \mathbb{R}$ and $\mathcal{Y} : \mathcal{B} \to \mathbb{R}$ are two functions such that $\mathcal{X} \geq_p \mathcal{Y}$, then:*

$$\boldsymbol{B}^{\pi}(\mathcal{X}) \geq_p \boldsymbol{B}^{\pi}(\mathcal{Y}) \tag{21}$$

Proof (Lemma). *For any b,*

$$\boldsymbol{B}^{\pi}(\mathcal{X})(b) - \boldsymbol{B}^{\pi}(\mathcal{Y})(b) = \sum_{s' \in \mathcal{S}} \mathcal{T}(s, \boldsymbol{ac}^{\pi}(b), s') \left[\gamma \sum_{o' \in \mathcal{O}} \mathcal{Z}(s', \boldsymbol{ac}^{\pi}(b), o') \left(\mathcal{X}(b_{\boldsymbol{ac}^{\pi}(b), o'}) - \mathcal{Y}(b_{\boldsymbol{ac}^{\pi}(b), o'}) \right) \right]$$

Since $\forall b' \in \mathcal{B}$, $\mathcal{X}(b') \geq \mathcal{Y}(b')$:

$$\forall o' \in \mathcal{O} \quad \mathcal{X}(b_{\boldsymbol{ac}^{\pi}(b), o'}) \geq \mathcal{Y}(b_{\boldsymbol{ac}^{\pi}(b), o'})$$

Also, since they represent probabilities, $\mathcal{T} \geq 0$ and $\mathcal{Z} \geq 0$. Thus,

$$\boldsymbol{B}^{\pi}(\mathcal{X})(b) - \boldsymbol{B}^{\pi}(\mathcal{Y})(b) \geq 0$$

Since $0 \leq \gamma < 1$, for any $\mathcal{X}$ and any π,

$$\lim_{l \to \infty} (\boldsymbol{B}^{\pi})^{l}(\mathcal{X})(b) = \mathcal{R}^{\pi}(b) \tag{22}$$

Thus, $(\boldsymbol{B}^{\pi})(\mathcal{R})(b) = \mathcal{R}^{\pi}(b)$.
Now, let $\boldsymbol{B}^{\pi_1} : (\mathcal{B} \to \mathbb{R}) \to (\mathcal{B} \to \mathbb{R})$ be the Bellman operator which is same as $\boldsymbol{B}^{\pi}$, except for $\forall b \in \mathcal{B}$ s.t. $\alpha_n = \arg\max_{\alpha \in \mathcal{V}^{\pi}} [b \bullet \alpha]$. For such b, $\boldsymbol{B}^{\pi_1}$ uses the expression of $\boldsymbol{B}^{\pi}$ with $\psi(\alpha_{n'})$ instead of $\boldsymbol{ac}^{\pi}(b)$. That is, the policy uses n' instead of n. By defining $\boldsymbol{ac}^{\pi_1}(b)$ appropriately, $\boldsymbol{B}^{\pi_1}$ can be written as:

$$\boldsymbol{B}^{\pi_1}(\mathcal{X})(b) = \sum_{s' \in \mathcal{S}} \mathcal{T}(s, \boldsymbol{ac}^{\pi_1}(b), s') \left[\mathcal{K}(s, \boldsymbol{ac}^{\pi_1}(b), s') + \gamma \sum_{o' \in \mathcal{O}} \mathcal{Z}(s', \boldsymbol{ac}^{\pi_1}(b), o') \mathcal{X}(b_{\boldsymbol{ac}^{\pi_1}(b), o'}) \right] \tag{23}$$

$\forall b \in \mathcal{B}$ s.t. $\boldsymbol{ac}^{\pi}(b) \neq \boldsymbol{ac}^{\pi_1}(b)$, due to equations 7, 8 and 18:

$$\boldsymbol{B}^{\pi_1}(\mathcal{R}^{\pi})(b) \geq \mathcal{R}^{\pi}(b) \tag{24}$$

Also, if for any b, $\boldsymbol{ac}^{\pi_1}(b) = \boldsymbol{ac}^{\pi}(b)$, then $\boldsymbol{B}^{\pi_1}(\mathcal{R}^{\pi})(b) = \mathcal{R}^{\pi}(b)$. Now, using the Lemma l-times with $l \to \infty$, for any b:

$$\lim_{l \to \infty} (\boldsymbol{B}^{\pi'})^{l}(\mathcal{R}^{\pi})(b) \geq \lim_{l \to \infty} (\boldsymbol{B}^{\pi})^{l}(\mathcal{R}^{\pi})(b) \tag{25}$$

Based on how $\boldsymbol{B}^{\pi_1}$ was defined, the LHS in inequality 25 actually represents the infinite horizon value function for a policy which is similar to π, except that node n' is chosen whenever n was supposed to be chosen in π. This is exactly the policy π'. Thus, $\lim_{l \to \infty} (\boldsymbol{B}^{\pi_1})^{l}(\mathcal{R}^{\pi})(b) = \mathcal{R}^{\pi'}(b)$ and,

$$\mathcal{R}^{\pi'}(b) \geq \mathcal{R}^{\pi}(b) \tag{26}$$

7 Subset Update

To reduce the size of π', we propose that only a random subset of $\mathcal{V}'$ be incorporated into π in the Policy Improvement step. From Section 6, it is clear that, as long as atleast one node is added to the controller, the property of policy improvement is guaranteed.

Algorithm 3: Modified Policy Improvement

```
SUBSET_POLICY_IMPROVEMENT ($\pi$, $bel$, $B$)
    Input: Initial controller $\pi$; Belief $bel$; Branching Factor $B$
    Result: Improved controller $\pi'$

    maxReward = $-\infty$;
    $\pi' \leftarrow \pi$;
    $(\mathcal{N}, \psi, \eta) \leftarrow \pi$;
    for $b \leftarrow 1$ to $B$ do
        $\pi_1 \leftarrow \pi$;
        get $\alpha$-vectors of $\pi$ as $\mathcal{V}$;
        $\mathcal{V}' \leftarrow DP\_update(\mathcal{V})$;
        subV $\leftarrow \phi$;
        forall $v \in \mathcal{V}'$ do
            //Choose uniformly at random from {0,1}
            x = boolRand(0.5);
            if $x = 1$ then
                subV $\leftarrow$ subV $\cup \{v\}$;
        end
        //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 $\mathcal{V}'$
        forall $\alpha_1 \in subV$ do
            if $\alpha_1 \in \mathcal{V}$ then
                continue;
            else
                add node representing $\alpha_1$ to $\pi_1$;
            end
        end
        forall $n$, $n' \in \pi_1$ s.t. $\alpha_n \geq_p \alpha_{n'}$ do
            redirect incoming edges of $n'$ in $\pi_1$ to $n$;
            delete $n'$ from $\pi_1$;
        end

        $\mathcal{V}_1 \leftarrow policyEvaluate(\pi_1)$;
        currReward $\leftarrow \max_{\alpha \in \mathcal{V}_1} [bel \bullet \alpha]$;
        if $currReward > maxReward$ then
            maxReward $\leftarrow$ currReward;
            $\pi' \leftarrow \pi_1$;
        end
    end
    return $\pi'$;
```

8 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,

$$n_0 = \arg\max_{n \in \mathcal{N}} [b_o \bullet \alpha_n] \tag{27}$$

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.

9 Experiments

9.1 Sample POMDPs

The following POMDP problems were used for evaluations:

9.1.1 Problem 1: Marketing Decisions [1]

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 $|\mathcal{S}| = |\mathcal{O}| = |\mathcal{A}| = 2$, $\gamma = 0.95$, $b_0 = (0.5)$. Refer to [1] for further details.

Actions	**Transition Probabilities**	**Observation Probabilities**	**Expected Reward**
Market Luxury Product (L)	$\begin{array}{c\|cc} & \mathbf{B} & \sim\mathbf{B} \\ \hline \mathbf{B} & 0.8 & 0.2 \\ \sim\mathbf{B} & 0.5 & 0.5 \end{array}$	$\begin{array}{c\|cc} & \mathbf{P} & \sim\mathbf{P} \\ \hline \mathbf{B} & 0.8 & 0.2 \\ \sim\mathbf{B} & 0.6 & 0.4 \end{array}$	$\begin{array}{cc} \mathbf{B} & \sim\mathbf{B} \\ \hline 4 & -4 \end{array}$
Market Standard Product (S)	$\begin{array}{c\|cc} & \mathbf{B} & \sim\mathbf{B} \\ \hline \mathbf{B} & 0.5 & 0.5 \\ \sim\mathbf{B} & 0.4 & 0.6 \end{array}$	$\begin{array}{c\|cc} & \mathbf{P} & \sim\mathbf{P} \\ \hline \mathbf{B} & 0.9 & 0.1 \\ \sim\mathbf{B} & 0.4 & 0.6 \end{array}$	$\begin{array}{cc} \mathbf{B} & \sim\mathbf{B} \\ \hline 0 & -3 \end{array}$

9.1.2 Problem 2: 4x3 Grid Navigation [2]

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 [2] for further details.

The Maze	**Rewards**	**States**	**Actions**	**Observations**	**Discount**
$\begin{array}{\|c\|c\|c\|c\|} \hline 0 & 0 & 0 & G \\ \hline 0 & 1 & 0 & T \\ \hline 0 & 0 & 0 & 0 \\ \hline \end{array}$	$\begin{array}{cc} \mathbf{G} & \mathbf{T} \\ \hline +1 & -1 \end{array}$	$\|\mathcal{S}\|$=11 Obstacle not counted	*NSEW*	*left, right, neither, both, good, bad, and absorb*	$\gamma = 0.95$

9.1.3 Problem 3: 4x3 Deterministic Grid [2]

Same as problem (9.1.2), except the underlying MDP is deterministic. That is, an action (N/S/E/W) will always have the intended result (p=1). If the underlying MDP is deterministic, then there exists a optimal policy.

9.1.4 Problem 4: 10x10 Grid Navigation [2]

A larger version of problem (9.1.2) so as to test scalability.

The Maze	**Rewards**	**States**	**Actions**	**Observations**	**Discount**
10x10 maze, 1 Goal state, 1 Trap State, 15 obstacles/walls	$\begin{array}{cc} \mathbf{G} & \mathbf{T} \\ \hline +1 & -1 \end{array}$	$\|\mathcal{S}\|$=85	*NSEW*	*left, right, neither, both, good, bad, and absorb*	$\gamma = 0.95$

9.2 Results

The following results were obtained upon using Subset Update, in conjunction with FSC Pruning (section 8)

Problem Number	**Better than Howard**	**Better than Howard by 15%**	**Better than Howard by 50%**
Problem 1, Node-Limit=100, Branching=4	Convergence Achieved	Convergence Achieved	Convergence Achieved
Problem 2, Node-Limit=100, Branching=8	8/10	8/10	6/10
Problem 3, Node-Limit=250, Branching=8	8/10	7/10	4/10
Problem 4, Node-Limit=500, Branching=16	9/10	7/10	5/10

10 Ongoing Work

10.1 Policy Iteration and Monte-Carlo Planning

Monte-Carlo Planning is guaranteed to improve upon a rollout policy in expectation. The Monte Carlo Tree Search algorithm does very well in practice even though it simply uses a random rollout policy. We can instead follow the policy given by the FSC for the rollouts.

10.2 Local Collapses In FSCs

In Hansen's algorithm, the nodes in π which are pointwise-dominated by any α-vector in $\mathcal{V}'$ are immediately deleted. This is done even before the policy evaluation step for π'. This action is valid because, when the old α-vector is replaced with the new one in the set of equations (7), the solution for all $V^{\pi}(n, o)$ can only converge to a higher value. This idea can be used to decrease the size of FSCs while also improving the policy.

- In each iteration, only those α-vectors from $\mathcal{V}'$ are added which dominate some node in π. Due to section (6), this step is valid.
- Now, based on the mentioned idea, all dominated vectors can immediately be deleted.
- Repeat the above two step until the FSC size is acceptable or is not decreasing further

References

[1] Eric A. Hansen. *An Improved Policy Iteration Algorithm for Partially Observable MDPs.* In Proceedings of the Conference on Neural Information Processing Systems, pages 1015–1021, Denver, CO, 1993.

[2] Eric A. Hansen. *Finite Memory Control of Partially Observable Systems.* Dissertation. University of Massachusetts Amherst., 1998.

[3] C. C. White and D. Harrington. *Application of Jensen's inequality for adaptive suboptimal design.* Journal of Optimization Theory and Applications, 32(1):89–99, 1980.

[4] George E. Monahan. *A survey of partially observable Markov decision processes: Theory, models and algorithms.* Management Science, 28:1–16, 1982.

[5] Michael L. Littman Anthony R. Cassandra and Nevin L. Zhang. *Incremental pruning: A simple, fast, exact method for POMDPs.* In Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence, pages 54–61, Providence, RI, 1997.

[6] Richard D. Smallwood and Edward J. Sondik. *The optimal control of partially observable Markov processes over a finite horizon.* Operations Research, 21:1071–1088, 1973.

[7] Edward J. Sondik. *The optimal control of partially observable Markov Decision Processes.* PhD thesis, Stanford university, Palo Alto, 1971.

[8] Hsien-Te Cheng. *Algorithms for Partially Observable Markov Decision Processes.* PhD thesis, University of British Columbia, Vancouver, 1988.

[9] Michael Littman Leslie Pack Kaelbling and Anthony R. Cassandra. *Planning and acting in partially observable stochastic domains.* Artificial Intelligence, 101:99–134, 1998.

[10] Ronald A. Howard. *Dynamic Programming and Markov Processes.* MIT Press, Cambridge, 1960.

[11] Joel Veness David Silver. *Monte-Carlo Planning in Large POMDPs.* Advances in Neural Information Processing Systems 23 (NIPS), 2010.

[12] Darius Braziunas. *POMDP solution methods.* Universität Bielefeld, 2003.

[13] C. Szepesvari. *Algorithms for Reinforcement Learning.* Morgan & Claypool, 2010.

[14] D. P. Bertsekas. *Dynamic Programming and Optimal Control.* volume 2. Athena Scientific, 4th edition, 2012.

[15] R. Bellman. *Dynamic Programming.* Princeton University Press, 1st edition, 1957.