Tractable Policy Iteration in POMDPs

Deep Karkhanis
deep_karkhanis@iitb.ac.in

Shivaram Kalyanakrishnan
shivaram@cse.iitb.ac.in

Bachelor's Thesis Project

IIT-Bombay
June 2020

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 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 are summarized in section 4. Section 5 describes a set of theorems which are necessary for the correctness of the algorithm called Subset Update that this text proposes in section 6. Section 7 mentions a way to reduce the FSC sizes during each iteration by a logical pruning step. Section 8 describes certain empirical results. Section 9 justifies the need of Policy Iteration for infinite horizon real-time problems. Section 11 specifies a few methods to combine two FSCs into 1. Section 10 highlights a intuitive way of assessing policy convergence by comparing POMDP policies with the underlying MDP's optimal policy. A few possibilities of future work have been highlighted in Section 12.

This text reports the entire work done as part of a B.Tech Project. Sections 1 through 6 were done as part of Stage I of the project and the rest were carried out during the Stage II of the project. However, as one might expect, certain inferences in the sections pertaining to the the first stage have been updated based on the work done during Stage II of the project. Also note that Section 8 has a few experiments which were carried out during the first stage of 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 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.

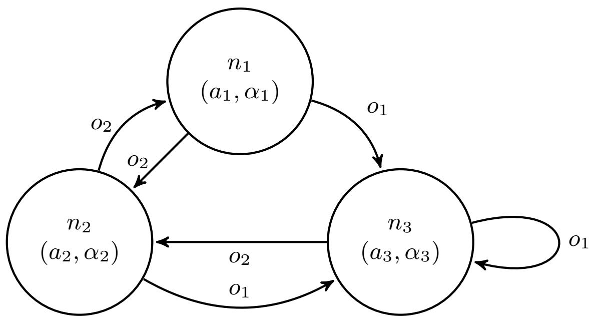


Figure 1: A sample FSC with $\mathcal{N} = \{n_1, n_2, n_3\}$. $\psi(n_i) = a_i$, $\forall i \in \{1, 2, 3\}$. Each node's successors for observations in $\mathcal{O} = \{o_1, o_2\}$ are denoted by edges. α_i is the α-vector corresponding to node n_i $\forall i \in \{1, 2, 3\}$.

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.

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 (Algorithm 1: 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 $\mathcal{V}_{t+1}$. In order to calculate $\mathcal{V}_{t+1}$, the DP-Update enumerates all possible actions and observation mappings to the set $\mathcal{V}_t$.

White and Harrington [2] 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 [3], and Incremental pruning [4]. 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 [5] and [6] 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 [7] 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 [8] 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.

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.

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 [9], 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' 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.

Policy Evaluation

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

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

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

$$V^\pi(n,s) = \alpha_n(s) \tag{8}$$

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

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 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 $\mathcal{V}'$

The optimal POMDP policy, if it exists is defined by the following optimality condition for its value function: [Sondik [6]] [Hansen [9]]

$$V^*(n,s) = \max_{a\in\mathcal{A}} \left[\sum_{s'\in\mathcal{S}} T(s,a,s')R(s,a,s') + \gamma \sum_{s'\in\mathcal{S}} \sum_{o\in\mathcal{O}} T(s,a,s')Z(s',a,o)V^*(\eta(n,o),s') \right] \tag{9}$$

More recently, the Point Based Policy Iteration (PBPI) algorithm was proposed by Shihao Ji et.al. [11]. Here, the authors concern with only a finite subset $B \subset \mathcal{B}$ of the belief space and iteratively find policies such that subsequent policies only ensure improvement for belief states in B. An improved policy may in fact decrease rewards for belief states not in B. Hence, PBPI might not be suitable in cases where the possible set of beliefs that an agent may encounter is not finite or hard to estimate.

4.3 Partially Observable Monte Carlo Planning

David and Joel [12] 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 [13] 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 Proof of Policy Improvement

Theorem 1. *If a controller π' can be obtained from a controller π by adding to π nodes which have no incoming connection, 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:*

$$\mathcal{R}'(b) \geq \mathcal{R}(b), \quad \forall\ b \in \mathcal{B} \tag{10}$$

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:*

$$\begin{aligned} &\mathcal{N} \subseteq \mathcal{N}' \\ &\psi(n) = \psi'(n), \forall n \in \mathcal{N} \\ &\eta(n,o) = \eta'(n,o), \forall n \in \mathcal{N}, \forall o \in \mathcal{O} \end{aligned} \tag{11}$$

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,

$$V^{\pi'}(n,s) = V^{\pi}(n,s) \ \forall n \in \mathcal{N}, s \in \mathcal{S} \tag{12}$$

Thus, using (4):

$$\alpha'_n = \alpha_n \ \forall n \in \mathcal{N} \tag{13}$$

Since $\mathcal{N} \subseteq \mathcal{N}'$ and using equation (13), $\forall\ b \in \mathcal{B}$:

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

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–*

$$\begin{aligned} \exists\ \alpha' \in \mathcal{V}'_1,\ b \in \mathcal{B}\ s.t. \quad [b \bullet \alpha'] &> \max_{n\in\mathcal{N}} [b \bullet \alpha_n] \\ thus, \quad \mathcal{R}'(b) &> \mathcal{R}(b) \end{aligned} \tag{14}$$

Note: $\mathcal{R}$ and $\mathcal{R}'$ are the reward functions as in Theorem (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'.

Lemma 1. *If a policy $\mathcal{V}$ is not optimal, DP-update($\mathcal{V}$) will give a better policy $\mathcal{V}'$*

Proof (Lemma). *Say $\widehat{\mathcal{V}'}$ is the set obtained in the DP-update, before pruning to a parsimonious set. Then, by definition, $\widehat{\mathcal{V}'}$ is the set of α-vectors which represents all possible first-step strategies available to the agent, given that it follows $\mathcal{V}$ 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 $\mathcal{V}$ themselves also represent valid strategies:*

$$\mathcal{V} \subseteq \widehat{\mathcal{V}'} \tag{15}$$

Thus, $\mathcal{V}'$ cannot worsen the expected reward for any belief state:

$$\forall b \in \mathcal{B} \ \max_{\alpha\in\mathcal{V}'} [b \bullet \alpha] \geq \max_{\alpha\in\mathcal{V}} [b \bullet \alpha] \tag{16}$$

Also, since $\mathcal{V}$ was already parsimonious,

$$\mathcal{V}' \not\subset \mathcal{V} \tag{17}$$

Assuming the method used to convert any set of α-vectors to a parsimonious set is deterministic, if $\mathcal{V} = \mathcal{V}'$ then $\mathcal{V}$ is an optimal policy since it satisfies the optimality equation 9. This is the case when policy iteration has converged.
If $\mathcal{V} \neq \mathcal{V}'$ then due to expression (17), $(\mathcal{V}' - \mathcal{V})$ is non-empty. Also, $\forall \alpha_o \in (\mathcal{V}' - \mathcal{V})$, since $\mathcal{V}'$ is parsimonious:

$$\exists b' \in \mathcal{B}\ s.t.\ \forall \alpha \in (\mathcal{V}' - \{\alpha_o\}) \quad [b' \bullet \alpha_o] > [b' \bullet \alpha] \tag{18}$$

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

$$\forall \alpha \in (\widehat{\mathcal{V}'} - \{\alpha_o\}) \quad [b' \bullet \alpha_o] \geq [b' \bullet \alpha] \tag{19}$$

According to expression (18), for some $b' \in \mathcal{B}$, only α_o gives the best expected reward, given that the agent follows FSC π second-step onwards. Moreover, there must be a continuous open region $\mathcal{D}_o$ in the belief space around b', where α_o 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 α_o and there would be another vector in $\mathcal{V}'$ (the vector sharing this boundary), which gives the same reward as α_o at b'. This would contradict expression (18). Also, since $\mathcal{D}_o$ is open, expressions (18) and (19) are true for every $b' \in \mathcal{D}_o$.

Now, if for any $\bar{\alpha} \in \widehat{\mathcal{V}}'$, the equality in expression (19) holds, then either $\exists b'' \neq b'$ in this continuous region $\mathcal{D}_o$ where α_o dominates $\bar{\alpha}$, or:

$$\alpha_o(s) = \bar{\alpha}(s) \ \forall s \in \mathcal{S} \tag{20}$$

This is because if $\bar{\alpha}$ matches the expected reward of α_o only for a finite number of $b \in \mathcal{D}_o$, then any other point in this infinite set $\mathcal{D}_o$ is a suitable b''. Moreover, the only way the reward can match for infinite number of $b \in \mathcal{D}_o$, is if the values of the α-vectors are same. Thus there are a finite number of such points for each α from the finite set $(\widehat{\mathcal{V}}' - \{\alpha_o\})$ which can give the same expected reward as α_o. Let $\mathcal{E}_o$ be the set of all such points. Since $\mathcal{D}_o$ is infinite, we can pick a $b \in (\mathcal{D}_o - \mathcal{E}_o$ for which expression (19) will be a strict inequality. That is,

$$\exists b \in \mathcal{B} \ s.t \ \forall \alpha \in (\widehat{\mathcal{V}}' - \{\alpha_o\}) \ \ [b \bullet \alpha_o] > [b \bullet \alpha] \tag{21}$$

Due to expression (15) and since $\alpha_o \in (\mathcal{V}' - \mathcal{V})$,

$$\exists b \in \mathcal{B} \ s.t \ \forall \alpha \in \mathcal{V} \quad [b \bullet \alpha_o] > [b \bullet \alpha] \tag{22}$$

Thus, proving the lemma.

Since $n' \notin \mathcal{N}$ i.e it is a new node, $\alpha_{n'} \notin \mathcal{V}$. Thus, using Lemma 1:

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

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:*

$$\alpha_{n'} \geq_p \alpha_n \tag{23}$$

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:

$$\forall b \in \mathcal{B}, \quad \mathcal{R}^{\pi'}(b) \geq \mathcal{R}^{\pi}(b) \tag{24}$$

Proof.
The proof is based on the proof of Policy Iteration for MDPs [14] and [15].
Let $\boldsymbol{B} : (\mathcal{B} \to \mathbb{R}) \to (\mathcal{B} \to \mathbb{R})$ be the equivalent of the Bellman operator [16] for POMDPs. For the FSC π, it is $\boldsymbol{B}^{\pi}$ s.t:

$$\boldsymbol{B}^{\pi}(\mathcal{X})(b) = \sum_{s' \in \mathcal{S}} \mathcal{T}(s, \boldsymbol{ac}^{\pi}(b), s') \left[\mathcal{K}(s, \boldsymbol{ac}^{\pi}(b), s') + \gamma \sum_{o' \in \mathcal{O}} \mathcal{Z}(s', \boldsymbol{ac}^{\pi}(b), o') \mathcal{X}(b_{\boldsymbol{ac}^{\pi}(b), o'}) \right] \tag{25}$$

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 2. *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:*

$$\boldsymbol{B}^{\pi}(\mathcal{X}) \geq_p \boldsymbol{B}^{\pi}(\mathcal{Y}) \tag{26}$$

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,

$$\lim_{l \to \infty} (\boldsymbol{B}^{\pi})^{l}(\mathcal{X})(b) = \mathcal{R}^{\pi}(b) \tag{27}$$

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:

$$\boldsymbol{B}^{\pi_1}(\mathcal{X})(b) = \sum_{s' \in \mathcal{S}} \mathcal{T}(s, \boldsymbol{ac}^{\pi_1}(b), s') \left[\mathcal{K}(s, \boldsymbol{ac}^{\pi_1}(b), s') + \gamma \sum_{o' \in \mathcal{O}} \mathcal{Z}(s', \boldsymbol{ac}^{\pi_1}(b), o') \mathcal{X}(b_{\boldsymbol{ac}^{\pi_1}(b),o'}) \right] \tag{28}$$

$\forall b \in \mathcal{B}$ s.t. $\boldsymbol{ac}^{\pi}(b) \neq \boldsymbol{ac}^{\pi_1}(b)$, due to equations 7, 8 and 23:

$$\boldsymbol{B}^{\pi_1}(\mathcal{R}^{\pi})(b) \geq \mathcal{R}^{\pi}(b) \tag{29}$$

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:

$$\lim_{l \to \infty} (\boldsymbol{B}^{\pi'})^{l}(\mathcal{R}^{\pi})(b) \geq \lim_{l \to \infty} (\boldsymbol{B}^{\pi})^{l}(\mathcal{R}^{\pi})(b) \tag{30}$$

Based on how $\boldsymbol{B}^{\pi_1}$ was defined, the LHS in inequality 30 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,

$$\mathcal{R}^{\pi'}(b) \geq \mathcal{R}^{\pi}(b) \tag{31}$$

6 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 5, it is clear that, as long as atleast one node is added to the controller, the property of policy improvement is guaranteed.

Algorithm 3: Subset Update Algorithm

```
Subset_Update_PI ($\pi_0$, $bel$, $B$, $\widehat{N}$)
    Input: Initial controller $\pi_0$; Initial Belief $bel$; Branching Factor $B$, Node Limit $\widehat{N}$
    Result: Improved FSC $\pi'$ with $|\mathcal{N}| \sim \widehat{N}$

    $\pi \leftarrow \pi_0$;
    while True do
        $\pi'$ , $conv$ ← SUBSET_POLICY_IMPROVEMENT($\pi$, $bel$, $B$, $\widehat{N}$);
        $(\mathcal{N}', \psi', \eta') \leftarrow \pi'$;
        if $|\mathcal{N}| = \widehat{N}$ or $\pi' = \pi$ or $conv = 1$ then
            return $\pi'$;
        $\pi \leftarrow \pi'$
    end
```

Algorithm 4: Modified Policy Improvement

```
SUBSET_POLICY_IMPROVEMENT ($\pi$, $bel$, $B$, $\widehat{N}$)
    Input: Initial controller $\pi$; Belief $bel$; Branching Factor $B$, Node Limit $\widehat{N}$
    Result: Improved controller $\pi'$, Boolean var whether $|\mathcal{N}| \sim \widehat{N}$

    maxReward = $-\infty$;
    $\pi' \leftarrow \pi$;
    $(\mathcal{N}, \psi, \eta) \leftarrow \pi$;
    //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 $\alpha$-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
            //$\alpha_1$ picked randomly each time without replacement
            if $\alpha_1 \in \mathcal{V}$ then
                continue;
            else
                if size of $\pi_1 \leq \widehat{N}$ then
                    add node representing $\alpha_1$ to $\pi_1$;
                else $conv_1 \leftarrow 1$ and break;
            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;
            $\pi' \leftarrow \pi_1$;
            $conv \leftarrow conv_1$;
        end
    end
    return $\pi'$, $conv$;
```

7 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,

$$n_0 = \arg\max_{n \in \mathcal{N}} [b_o \bullet \alpha_n] \tag{32}$$

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.

8 Experiments

8.1 Sample POMDPs

The following POMDP problems were used for evaluations:

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

Actions	**Transition Probabilities**			**Observation Probabilities**			**Expected Reward**	
		B	**∼B**		**P**	**∼P**	**B**	**∼B**
Market Luxury Product (L)	**B**	0.8	0.2	**B**	0.8	0.2	4	-4
	∼B	0.5	0.5	**∼B**	0.6	0.4		
		B	**∼B**		**P**	**∼P**	**B**	**∼B**
Market Standard Product (S)	**B**	0.5	0.5	**B**	0.9	0.1	0	-3
	∼B	0.4	0.6	**∼B**	0.4	0.6		

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 [9] 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: 4x3 Deterministic Grid

Same as problem (8.1.2), 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.

8.1.4 Problem 4: 10x10 Grid Navigation

A larger version of problem (8.1.2) so as to test scalability.

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

8.2 Results

The following results were obtained upon using Subset Update (Algorithm 3). The inputs Branching Factor B and Node Limit $\widehat{N}$ have been mentioned in the first column of the table. 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 (probability of being in the goal or trap is 0 initially)

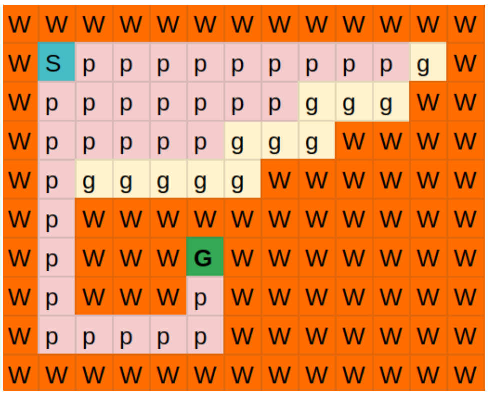
Problem Number	**Better than Howard**	**Better than Howard by 15%**	**Better than Howard by 50%**
Problem 1, Node-Limit=100, Branching=4	Convergence Achieved	Convergence Achieved	Convergence Achieved
Problem 2, Node-Limit=100, Branching=8	8/10	8/10	6/10
Problem 3, Node-Limit=250, Branching=8	8/10	7/10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

9 Comparing Policy Iteration with other algorithms

The primary competitors to Policy Iteration (PI) are Monte-Carlo (MC) Planning and Value Iteration. As mentioned earlier, value iteration does not guarantee infinite horizon policies unless it converges and hence is not appropriate for infinite horizon problems. Monte-Carlo Planning is a online algorithm i.e the planning and computation is being done while the agent is moving through the environment. Thus, it is not suitable for complex real-time problems. Moreover, it can also be the case that the rollouts are not deep enough to identify potential high reward path. This can be seen in the following example where Monte Carlo Planning was not able to identify the low probability high reward goal within reasonable amount of computation time unlike Policy Iteration.

9.1 Multiple Reward Maze

In maze 2, there is a high reward hidden which can only be reached by taking a single path. Multiple small rewards are available and easily reachable in the maze. A time limit has been set to the amount of computations the algorithm can perform during each online step. This has been done to mimic requirements for real-time problems. Both value iteration and Monte-Carlo planning are mislead into choosing the cheaper paths and get lower rewards. Policy Iteration however is capable of reaching the higher reward. Since it is an offline algorithm, allowing Policy Iteration to run for a larger amount of time is reasonable. Monte-Carlo planning not being able to identify the high reward could be attributed to the limited computation time allowed in the online setting, and the pruning of the search tree done at each step. This might have lead to the search ”overlooking” the higher reward.


W W W W W W W W W W W W W
W S p p p p p p p p p g W
W p p p p p p p g g g W W
W p p p p p g g g W W W W
W p g g g g g W W W W W W
W p W W W W W W W W W W W
W p W W W G W W W W W W W
W p W W W p W W W W W W W
W p p p p p W W W W W W W
W W W W W W W W W W W W W

Rewards:

g	**p**	**G**
+1	-1	+1000

Actions, states and observations are as in 8.1.2
W indicate walls or unreachable states

Experiments (20 runs each):

Algorithm	Found g	Found G	No reward
MCTS	15/20	5/20	Never
PI	4/20	16/20	Never

Average rewards
for PI: 762.86 for MCTS: 287.11

Figure 2: Maze with multiple rewards

When MCTS algorithm was used for the multiple rewards maze figure (2), a time limit of 0.1s was set for each step of simulation for the maze (On HP-Pavilion with intel i5 Gen6 processor). The MCTS branching factor and expansion depth was set to 4 to achieve this time limit (note: random rollouts may reach a much higher depth). For the Policy Iteration method, the node limit was set to 1000. With the final FSCs having node sizes between 992 and 1000.

10 Achieving MDP policies

In the algorithms discussed, the convergence was based on a node limit set apriori. However, such a parameter does not immediately indicate the convergence and goodness of POMDP policies. It seems that to come up with a good node limit, one ultimately must actually simulate the usage of the FSC itself. This section tries to address this concern.

POMDPs have an underlying MDP. It is much easier and faster to compute the optimal infinite horizon policy for the MDP. Comparing intermediate POMDP policies with the optimal MDP policy can thus be useful. Policy Iteration improves the policy in each iteration. Since a POMDP agent actually moves through the MDP itself, the expected rewards for the POMDP policy will be bound by the optimal MDP policy. A plot of expected rewards v/s MDP states can thus be useful in deciding when to terminate policy iteration. The plot in figure 3 has been plotted for a maze similar to problem 8.1.2 when run on the Subset Update Algorithm (3). The POMDP policy (FSC) generated after every 5th iteration was compared with the optimal MDP policy. This was done by finding the POMDP policy's expected reward for every deterministic belief state (i.e any state b which has $b(s) = 1.0$ for exactly one MDP state s). As can be seen iteration 20 gives values very close to iteration 15. One can thus use the sum squared distance as a metric to come up with a convergence strategy.

Also, potentially building $|S|$ number of FSCs each providing high expected reward for a different state, and them combining them could lead to a single very good FSC which represents close to optimal policy.

11 Combining FSCs

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.

11.1 Union FSC

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 $\widehat{\pi}$ which has two disconnected components, each corresponding to one of the original FSCs. Although such a formulation does give a

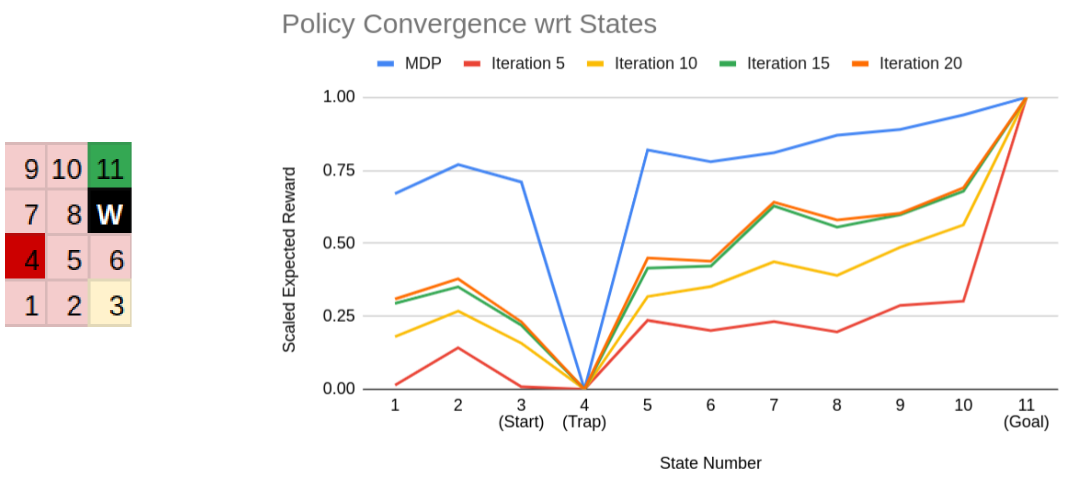


Figure 3: Variant of the 4x3 Maze 8.1.2. Values in grid-cells are state numbers of the underlying MDP

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. In the combined FSC (named union FSC) suggested earlier, the agent will stay in only one of the disconnected components after the first step. Thus, even though the combined FSC will do as good as the individual FSCs, the improvement can only be expected to be marginal.

11.2 Belief State Tracking

A better way to use the combined FSC can be by simply keeping track of the belief state (Algorithm 5) while using the combined FSC generated in 11.1.

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.

Algorithm 5: Using Combined-FSC by tracking belief

```
BELIEF_TRACKING ($o$)
    Input: Last observation $o$
    Result: Yields next action $a$ on every call
    Global Variables Initial belief $bel$; Union of Nodes $\mathcal{N} = \mathcal{N}_1 \cup \mathcal{N}_2$; Combined Action
     Function $\psi$

    // First call doesn't need an observation
    $b \leftarrow bel$;
    $nd \leftarrow \arg\max_{n \in \mathcal{N}} [b \bullet \alpha_n]$;
    yield $\psi(nd)$;
    while True do
        // Subsequent calls need previous observations
        $b \leftarrow update\_bel(b, \psi(nd), o)$;
        $nd \leftarrow \arg\max_{n \in \mathcal{N}} [b \bullet \alpha_n]$;
        yield $\psi(nd)$;
    end
```

11.3 Splitting Nodes

In the above mentioned method each step requires recomputing the belief state and the minimum expected rewards provided by each node. In real-time applications, such a approach is not desirable. One of the important advantages of using an FSC is that when the policy is been used, one does not keep track of the belief or compute the next best node but still gets the expected reward computed from the dot-product of the node where one started. Thus, to ensure the notion of Policy Iteration (looping each time the same situation arises/combined FSC has no disconnected component) we would ideally like a combined FSC which promises the same expected rewards as in section 11.2 but without the need to compute belief state each time (i.e simply following the successor is enough to ensure the expected reward)

However, we believe that such an FSC might not exist if the number of nodes are to be kept finite. This is due to the fact that the expected reward promised by a node is a function of its successors' α-vectors. It specifies an action and a subsequent node for every resulting observation. Say a new set of nodes (i.e a new FSC) is to be added to an existing FSC such that the rewards are as promised in 11.2. It might happen that for belief states which are promised maximum reward in a node n would need n to have a successor in the original FSC to ensure this. (i.e subsequent nodes that n has for each of its observations might not be the best node for every belief state which uses n as a start node). Thus we must add a new nodes which represent these best possible choices, or in other words, split the node n. Thus, we are increasing our set of new nodes to be added and the process may never end. The important thing to note here is that the goal is to split nodes so that we might get as close to 11.2 as possible.

We can thus, try to solve a easier problem. That is, instead of trying to split all nodes, we simply find all splits of a particular node and let all other nodes remain untouched. Let $\mathcal{N} = \mathcal{N}_1 \cup \mathcal{N}_2$ is set of all nodes from the two FSCs and say we performed a DP-update 1 on $\mathcal{N}$ and added those nodes (acc. to Hansen's algorithm 2) to the combined FSC $\widehat{\pi}$ (FSC as stated earlier). Then since DP-update does a one step look ahead, it will exhaustively give us the best possible choices that various belief states might need if they follow FSC $\widehat{\pi}$ second step onwards. Thus, they would also include strategies where splits of any but exactly one node might have been made since the possibility of successors would be the same. Now, once these vectors are added to $\widehat{\pi}$, we have solved our easier problem.

Figure 4 highlights the motivation behind combining FSCs. FSCs with node limit of 200 were generated for problem 8.1.2 using the subset update algorithm (Algorithm 3) and then combined to get the union FSC. The average reward over multiple runs have been plotted. Simply using the union FSC does not give much improvement as more and more FSCs are combined. Thus is because since the initial belief state is known, in practice only one of the component FSCs will ever be used.

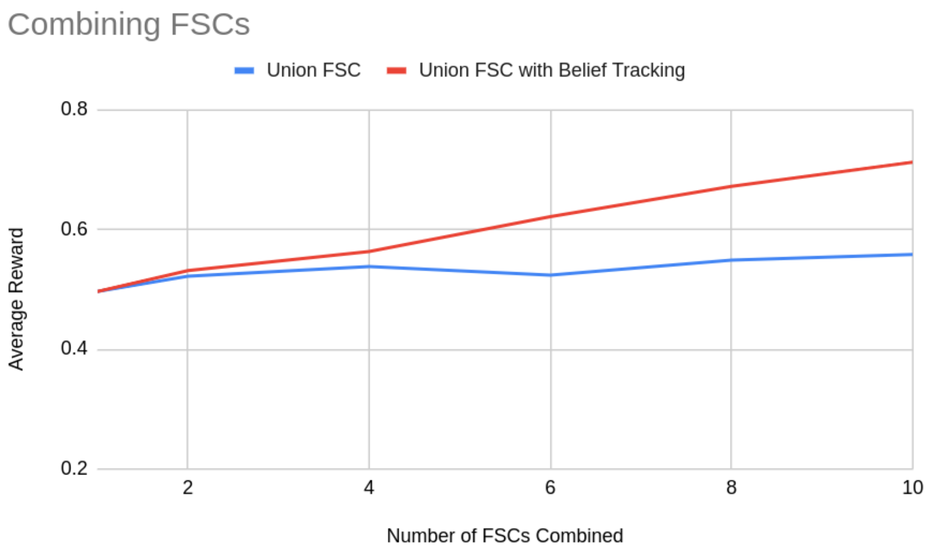


Figure 4: Increasing rewards achieved by combining FSCs

12 Future Work

This text has attempted to reignite the importance and potential advantages of pursuing research in POMDP Policy Iteration. Multiple methods have been proposed to improve upon the existing methods. It is important to note that for real-time infinite horizon problems, Policy Iteration might be the most promising method thanks to offline computations and infinite horizon policies. Further work can be done to find more problems like the multiple-rewards maze 9.1, which are closer to the real-world while also being real-time and infinite horizon. A worthy candidate is the packet-routing problem at an intermediate node in the internet. At each step, it must be quickly decided which one of the multiple paths should be taken in order to route a packet depending on varied congestion in them. Apart from this, the analysis of POMDP policies based on expected rewards in various start states (section 10) can also be explored 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] 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.

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

[4] 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.

[5] 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.

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

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

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

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

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

[11] Hui Li Xuejun Liao Shihao Ji, Ronald Parr and Lawrence Carin. *Point Based Policy Iteration.* AAA1, 2007.

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

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

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

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

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