Graduate School ETD Form 9
(Revised 12/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Mohit Sachan

Entitled
Learning in Partially Observable Markov Decision Processes

For the degree of Master of Science

Is approved by the final examining committee:

Snehasis Mukhopadhyay
Chair

Rajeev Raje

Mohammad Al Hasan

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Snehasis Mukhopadhyay

Approved by: Shiaofen Fang 07/02/2012
Head of the Graduate Program Date

Graduate School Form 20
(Revised 9/10)

PURDUE UNIVERSITY
GRADUATE SCHOOL

Research Integrity and Copyright Disclaimer

Title of Thesis/Dissertation:

Learning in Partially Observable Markov Decision Processes

For the degree of Master of Science

I certify that in the preparation of this thesis, I have observed the provisions of *Purdue University Executive Memorandum No. C-22,* September 6, 1991, *Policy on Integrity in Research.**

Further, I certify that this work is free of plagiarism and all materials appearing in this thesis/dissertation have been properly quoted and attributed.

I certify that all copyrighted material incorporated into this thesis/dissertation is in compliance with the United States' copyright law and that I have received written permission from the copyright owners for my use of their work, which is beyond the scope of the law. I agree to indemnify and save harmless Purdue University from any and all claims that may be asserted or that may arise from any copyright violation.

Mohit Sachan

Printed Name and Signature of Candidate

07/02/2012

Date (month/day/year)

*Located at http://www.purdue.edu/policies/pages/teach_res_outreach/c_22.html

LEARNING IN

PARTIALLY OBSERVABLE MARKOV DECISION PROCESSES

A Thesis

Submitted to the Faculty

of

Purdue University

by

Mohit Sachan

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

August 2012

Purdue University

Indianapolis, Indiana

This work is dedicated to my family and friends.

ACKNOWLEDGMENTS

I am heartily thankful to my supervisor, Dr. Snehasis Mukhopadhyay, whose encouragement, guidance and support from the initial to the final level enabled me to develop an understanding of the subject. He patiently provided the vision, encouragement and advise necessary for me to proceed through the masters program and complete my thesis.

Special thanks to my committee, Dr. Rajeev Raje and Dr. Mohammad Al Hasan for their support, guidance and helpful suggestions. Their guidance has served me well and I owe them my heartfelt appreciation.

Thank you to all my friends and well-wishers for their good wishes and support. And most importantly, I would like to thank my family for their unconditional love and support.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Sachan, Mohit. M.S., Purdue University, August 2012. Learning in Partially Observable Markov Decision Processes. Major Professor: Snehasis Mukhopadhyay.

Learning in Partially Observable Markov Decision process (POMDP) is motivated by the essential need to address a number of realistic problems. A number of methods exist for learning in POMDPs, but learning with limited amount of information about the model of POMDP remains a highly anticipated feature. Learning with minimal information is desirable in complex systems as methods requiring complete information among decision makers are impractical in complex systems due to increase of problem dimensionality.

In this thesis we address the problem of decentralized control of POMDPs with unknown transition probabilities and reward. We suggest learning in POMDP using a tree based approach. States of the POMDP are guessed using this tree. Each node in the tree has an automaton in it and acts as a decentralized decision maker for the POMDP. The start state of POMDP is known as the landmark state. Each automaton in the tree uses a simple learning scheme to update its action choice and requires minimal information. The principal result derived is that, without proper knowledge of transition probabilities and rewards, the automata tree of decision makers will converge to a set of actions that maximizes the long term expected reward per unit time obtained by the system. The analysis is based on learning in sequential stochastic games and properties of ergodic Markov chains. Simulation results are presented to compare the long term rewards of the system under different decision control algorithms.

1 INTRODUCTION

A Markov chain is a mathematical system that undergoes transitions from one state to another, between a finite or countable number of possible states. It is a random process characterized as memoryless: the next state depends only on the current state and not on the sequence of events that preceded it. This specific kind of "memorylessness" is called the Markov property [1].

Following is an example of Markov chain.

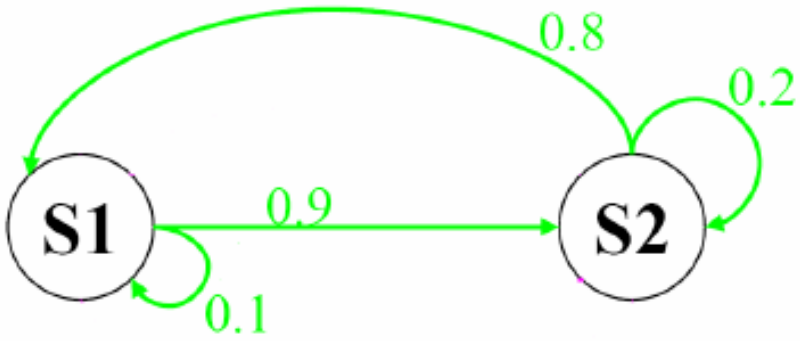

Figure 1.1. Markov Process Example

In Figure 1.1 there are 2 states $S1$ and $S2$ in Markov chain. The agent makes a transition from $S1$ to $S2$ with probability $p = 0.9$ and remain in the same state $S1$ with probability $p = 0.1$. Similarly when in state $S2$, it makes transition to state $S1$ with probability 0.8 and remains in same state $S2$ with probability $p = 0.2$. The transition to the next state depends only on the current state of the agent.

If we add uncertainty to a markov chain in the form that we cannot see what state we are currently in we get a hidden markov model (HMM). In a regular Markov model, the state is directly visible to the observer, and therefore the state transition probabilities are the only parameters whereas in a HMM, the state is not directly visible, but output, dependent on the state, is visible. Each state has a probability distribution over the possible output observations. Therefore the sequence of observations generated by an HMM gives some information about the sequence of states. HMM are especially known for their application in pattern recognition such as speech, handwriting [2], gesture recognition [3], part-of-speech tagging [4], musical score following [5], partial discharges [6] and bioinformatics.

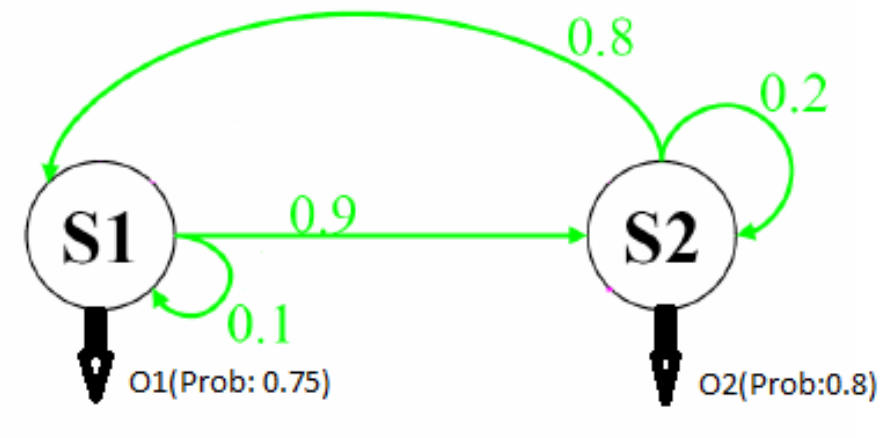

Figure 1.2. Hidden Markov Model Example

In Figure 1.2 of Hidden Markov Model, we have two states $S1$ and $S2$. The agent makes a transition from $S1$ to $S2$ with probability $p = 0.9$ and remains in the same state $S1$ with probability $p = 0.1$. Similarly from state $S2$, the agent makes a transition to state $S1$ with probability 0.8 and remains in same state $S2$ with probability $p = 0.2$. But the states are not visible to the agent directly, instead it sees an observation symbol $O1$ with probability $p = 0.75$ when it is in state $S1$ and observation symbol $O2$ with probability $p = 0.8$ when in state $S2$.

Addition of controllable actions in each state in a Markov chain gives us a Markov Decision Process (MDP). In MDP, the next state is determined by the current state and an action. The Markov Property holds for MDP also as it is memoryless and depends only on current state and current action. Markov Decision Processes are an extension of Markov chains; the difference being the addition of actions in each state (allowing choices) and the assignment of rewards or penalty for taking an action (adding motivation) in each state. If there is only one action available for each state and all rewards are zero, a Markov decision process reduces to a Markov chain.

Following is an example of Markov Decision Process.

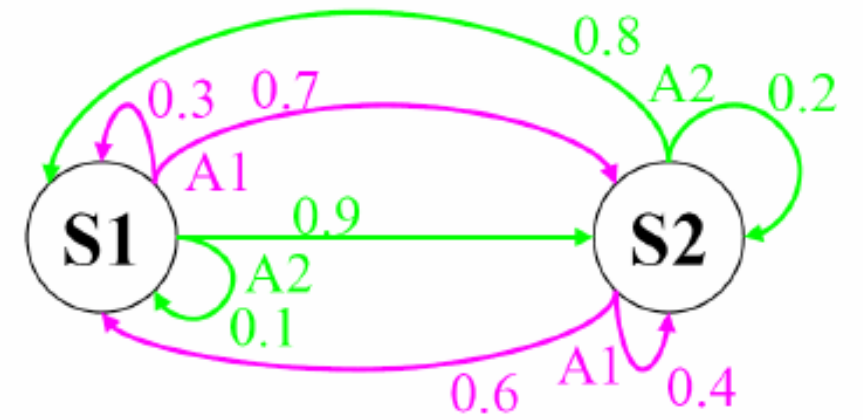

Figure 1.3. Markov Decision Process Example

In Figure 1.3 of MDP, we have two states $S1$ and $S2$. There are two actions $A1$ and $A2$ available in each of the states. In state $S1$ if agent takes action $A1$, it moves to state $S2$ with probability $p = 0.7$ and if it takes action $A2$ it moves to state $S2$ with probability $p = 0.9$. Similarly at state $S2$ the agent moves to state $S1$ with probability $p = 0.6$ if it takes action $A1$ and it moves to $S1$ with probability $p = 0.8$ if it takes action $A2$.

Further introduction of uncertainty in Markov Decision Processes gives rise to Partially Observable Markov Decision Processes (POMDPs). In POMDP we cannot see which state we are currently in, however each state emits observation symbols. Thus the only way to guess the present state is through the emitted observation symbols.

Following is an example of a Partially Observable Markov Decision Process.

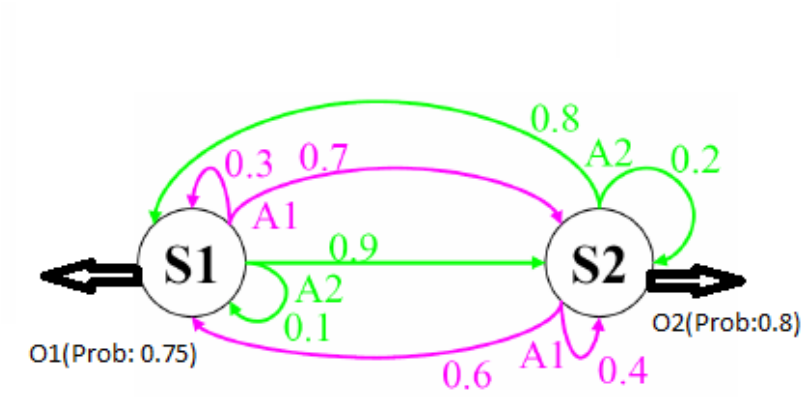

Figure 1.4. Partially Observable Markov Decision Process Example

The POMDP in Figure 1.4 contains two states $S1$ and $S2$. Each state has choice of two actions $A1$ and $A2$ available. The agent moves from $S1$ to $S2$ with probability $p = 0.7$, if it chooses action $A1$ and if it choose $A2$ at $S1$ it moves to $S2$ with probability $p = 0.9$. Similarly at state $S2$ the agent moves to state $S1$ with probability $p = 0.6$, if it takes action $A1$ and moves to state $S1$ with probability $p = 0.8$, if it takes action $A2$.

The agent does not see which state it is in instead it see the observation symbol emitted by the states. State $S1$ emits observation symbol $O1$ with probability $p = 0.75$ and state $S2$ emits observation symbol $O2$ with probability $p = 0.8$.

The following diagram outlines the difference between different Markov models.

Markov Models		Do we have control over the state transitons?	
		NO	YES
Are the states completely observable?	YES	**Markov Chain**	**MDP** Markov Decision Process
	NO	**HMM** Hidden Markov Model	**POMDP** Partially Observable Markov Decision Process

Figure 1.5. Comparison of different Markov models

In real life, decisions that humans and computers make on all levels usually have two types of impacts:

- They cost or save time, money, or other resources, or they bring revenues,
- They have an impact on the future, by influencing the dynamics.

In many situations, decisions with the largest immediate profit may not be good in view of future events. MDPs model this paradigm and provide results on the structure and existence of good policies and on methods for their calculation. MDPs have attracted the attention of many researchers because they are important both from the practical and the intellectual point of view. MDPs provide tools for the solution of important real-life problems [7] and give a mathematical framework for modeling

decision-making in situations where outcomes are partly random and partly under the control of a decision maker. They have proven to be useful particularly in a variety of sequential planning applications where it is crucial to account for uncertainty in the process [8].

A Markov decision process (MDP) is defined by the following 4 elements:

- A finite number of states of the environment $\Phi = \{\phi_1, \phi_2, ..., \phi_n\}$,
- A finite set of actions available $\alpha = \{\alpha_1, \alpha_2, ..., \alpha_k\}$
  (Alternatively, α_i is the finite set of actions available from state ϕ_i.),
- A payoff function
  $R : \Phi \times \Phi \times \alpha \to \{-1, 0, 1\}$
  such that $r^i_j(k) = R(\phi_i, \phi_j, \alpha)$ is the immediate reward (or expected immediate reward) received after transition from state ϕ_i to state ϕ_j using action α. (Here -1 corresponds to penalty, 0 corresponds to no feedback, 1 corresponds to reward),
- A state transition probability function
  $P : \Phi \times \Phi \times \alpha \to (0, 1)$
  Where $p^i_j = P(\phi_i, \phi_j, \alpha)$ determines the probability that action α in state ϕ_i at time t will lead to state ϕ_j at time $t + 1$. $P_\alpha(\phi_i, \phi_j) = Pr(\phi_{t+1} = \phi_j | \phi_t = \phi_i, \alpha_t = \alpha)$.

The objective of learning algorithm in the MDP is to determine a policy $\pi : \Phi \to \alpha$ which results in maximum long term reward.

A Partially Observable Markov Decision Process (POMDP) is further generalization of a Markov Processes. A POMDP is similar to an MDP; we have a set of states, a set of actions, and transition among states and finally get rewards as effect of transition. The actions effect on the state in a POMDP is exactly the same as in an MDP. The

difference being we can't observe the current state of the process so in a POMDP we add a set of observations to the model. Now instead of directly observing the current state, the state gives us an observation token, which provides a hint about the state in which the process may reside. These observations are generally probabilistic; so we need to also specify an observation function. This observation function tells us the probability of each observation for each state in the model. The observation likelihood can also be made to depend on the action if needed.

An Agent in artificial intelligence (AI) is a system that perceives its environment and takes action that maximizes its chance of success. One of the goals of AI is to design an agent which can interact with an environment so as to maximize some reward function.

A POMDP models an agent decision process where system dynamics are determined by an MDP, but the agent cannot directly observe the underlying state. To know what state it is in, it maintains a probability distribution over the set of possible states, based on a set of observations and observation probabilities, and the underlying MDP. The POMDP framework is a general framework and it can model a variety of real-world sequential decision processes. This model augments a well-researched framework of Markov decision processes (MDPs) [8], [9] to situations where an agent cannot reliably identify the underlying environment state. The POMDP formalism is very general and powerful, extending the application of MDPs to many realistic problems [10].

A POMDP is defined as a tuple $(\Phi, \alpha, O, T, \Omega, R)$, where

- S is a set of states $\Phi = \{\phi_1, \phi_2, ..., \phi_n\}$,
- A is a set of actions $\alpha = \{\alpha_1, \alpha_2, ..., \alpha_k\}$,
- O is a set of observations $O = \{o_1, o_2, ..., o_m\}$,

- T is a set of conditional transition probabilities, $P(\phi_j|\phi_i, \alpha)$,
- Ω is a set of conditional observation probabilities $P(O|\Phi)$,
- $R : \Phi \times \Phi \times \alpha \to \{-1, 0, 1\}$ is the reward function.

At each time period, the environment is in some state $\phi_i \in \Phi$. The agent takes an action $\alpha^i_{r_i} \in \alpha$, which causes the environment to transition to state ϕ_j with probability $T(\phi_j|\phi_i, \alpha^i_{r_i})$. Finally, the agent receives a reward with expected value, say $r^i_j(k)$, and the process repeats. The difficulty is that the agent does not know the exact state it is in. Instead, it must maintain a probability distribution, known as the belief state, over the possible states Φ. An agent needs to update its belief upon taking the action α and observing O. Since the state is Markovian, maintaining a belief over the states solely requires knowledge of the previous belief state, the action taken, and the current observation. The operation is denoted $b\prime = \tau(b, \alpha_i, o)$. Where $b\prime$ is the present belief state and it depends on previous belief state b, action taken α_i and the observation symbol o seen in the transition.

POMDP problems with various performance criteria have been posed and the uses of dynamic programming methods to determine the optimal policy are well known [9], [11]. However, several important factors have limited the applicability of this type of approach. First, the computation becomes burdensome when the number of states is large. Secondly, There is no way to guess the current state based on observation symbol with certainty. Third, the information about the model that is required for an approach such as dynamic programming is not available. Specifically, transition probabilities and corresponding rewards associated with various actions may be unknown at the time control begun or may change during system operation. This leads to a new adaptive problem in which, typically, parameters are estimated and, using a separation principle, the subsequent estimates are used to update control actions [12], [13].

Due to the generality of POMDPs, it entails a high computational cost to solve it. The problem of finding optimal policies for finite-horizon POMDPs has been proven to be PSPACE-complete [14]. Because of the intractability of current solution algorithms, especially those that use dynamic programming to construct (approximately) optimal value functions [15], [16], the application of POMDPs remains limited to very small problems.

We suggest a method that addresses learning problem is POMDP and avoids a lot of computational difficulty. The approach is different from many other currently used approaches as no dependence on an unknown parameter is assumed. We suggest a learning approach based on a tree, in which each node contains a learning automaton (LA). The root node of the tree is the only known start state (Landmark state) of POMDP. Each node in the tree has child nodes corresponding to actions available and observation symbol emitted by the states. Each node in this LA tree corresponds to a POMDP state. Each state in POMDP chooses its action through a corresponding LA in the tree independently without the knowledge of outer world. There is no knowledge that other agents exist or indeed that the world is an N-state POMDP whose transition probabilities and corresponding rewards depend on actions chosen. Each LA in the tree tries to improve its own performance by choosing a favorable action. It chooses an action and waits for a response. No information is passes until the process returns to the same node again. Once the process returns to the same node the LA receives the required information and updates its action.

There is no need for explicit synchronization of different LAs in the tree. Action at each LA node is updated only when the process returns to the same state. The updating is done via a simple learning scheme. This scheme uses a cumulative reward obtained from a given action normalized by the total elapsed time under that action as its environmental response.

The result is that individuals operating in nearly total ignorance of their surroundings can implicitly coordinate themselves to lead to optimal group behavior. This result is based on a result on learning in N-player identical payoff games [17].

The Landmark based approach is practical and not limiting because in most POMDP problems we have information available about the starting states and starting state may have some sensor that will make sure about the state when the process returns to this state again. The Landmark state relies on the availability of sensor information to make sure of the state.

1.1 Organization of thesis

The thesis is broadly divided into two parts: state estimation in POMDP using a tree and then learning in POMDP using that state estimation tree. Chapter 2 will discuss the background necessary for understanding this thesis and some current solution to POMDP problems. In chapter 3, state estimation will be discussed in detail. We will describe how a state in POMDP corresponds to a node in our tree. Chapter 4 will discuss the learning algorithm using learning automata in state estimation tree in details and how each node in the tree updates its actions. Chapter 5 will show the results and simulation of the learning algorithm on a simple POMDP problem and will compare the results with other algorithms. Chapter 6 concludes our work and suggests future work.

2 BACKGROUND LITERATURE

This thesis draws motivation from the work done by Richard M. Wheeler and Kumpati S. Narendra which describes method for decentralized learning in Markov Decision Processes [17]. This thesis takes similar approach for learning in POMDP. In [17], they address adaptive problem in MDP where transition probabilities may change during the MDP process. [17] suggests model setting of myopic local agents, one located at each state of MDP, which is unaware of the surrounding world. There is no knowledge that other agents exist or indeed that the world is an N- state Markov chain whose transition probabilities and corresponding rewards depend on actions chosen. The approach works well for MDPs but cannot be used when there is uncertainty in states as we dont know what state we are currently in.

A policy is a set of rules that define what action to take in what state in an MDP or POMDP such that long term rewards are maximized. In order to find a policy or decision control in POMDP we need some form of memory for our agent to choose actions correctly [18]. We need to maintain a probability distribution over the states of underlying environment. This distribution is called belief state and is normally represented as $b(s)$ to indicate what agent believes about its current state. Using the POMDP model the belief states are updated based on the agents action and observations such that the belief states correspond exactly to the state occupation probabilities. Since the agents belief state is an accurate summary of all relevant past information it can be used by agent to choose optimal action. Belief states in combination with the updating rule form a completely observable MDP with a continuous state space.

The agent's policy π specifies an action $\alpha = \pi(b)$ for any belief b. The optimal policy π^* yields the highest expected reward value for each belief state and is represented by optimal value function V^*. A powerful result of [15] is that optimal value function for any POMDP can be approximated arbitrarily well by a piecewise linear and convex function (PWLC). There exist a class of POMDP that has a value function exactly as PWLC [15]. These results apply to optimal Q function, where Q function for action α, $Q_\alpha(b)$ is the expected reward for a policy. For the Q function $Q_\alpha(b)$ the policy takes action α in belief state b and behaves optimally. To behave optimally, the agent chooses an action α that has the largest Q value for the given belief state. The representation simplicity of PWLC functions makes them convenient. A PWLC function $Q_\alpha(b)$ can be written simply as

$$Q_\alpha(b) = \max_{q \in L_\alpha} q.b \tag{2.1}$$

where L_α is a finite set of S dimensional vector. So $Q_\alpha(b)$ is the maximum of a finite set of linear functions of b. To solve a POMDP using Q function we can temporarily ignore the observation model and make use of the Q values of the underlying MDP.

Some of the methods used to solve POMDP are discussed in the following sections.

2.1 POMDP value iteration

Value iteration for MDPs is a standard method of maximizing long term reward and finding the optimal infinite horizon policy π^* using a sequence of optimal finite horizon value functions $V_0*, V_1*, V_2 * ..V_t*$ [9]. The difference between the optimal value function and the optimal t-horizon value function goes to zero as t goes to infinity:

$$\lim_{t \to \infty} \max_{s \in S} \mid V^*(s) - V_t^*(s) \mid = 0. \tag{2.2}$$

Any POMDP can be reduced to a continuous belief-state MDP. Therefore, value iteration can also be used to calculate optimal infinite horizon POMDP policies as following:

- Initialize $t = 0$ and $V_0(b) = 0$ for all $b \in B$
- While $\max_{b \in B} \mid V_{t+1}(b) - V_t(b) \mid > \epsilon$, calculate $V_{t+1}(b)$ for all states $b \in B$ according to the following equation, and then increment t:

$$V_{t+1}(b) = \max_{\alpha_b \in \alpha} \left[R^b(b, \alpha_b) + \gamma \sum_{b\prime \in B} T^b(b, \alpha_b, b\prime) V_t(b\prime) \right] \tag{2.3}$$

  where γ is discount factor.

Although the belief space is continuous, any optimal finite horizon value function is piecewise linear and convex and can be represented as a finite set of $\alpha-$vectors [10]. Therefore, the essential task of all value-iteration POMDP algorithms is to find the set V_{t+1} representing value function V_{t+1}, given the previous set of $\alpha-$vectors V_t

Various POMDP algorithms differ in how they compute value function representations. The most naive way is to construct the set of conditional plans V_{t+1} by enumerating all the possible actions and observation mappings to the set V_t. Since many vectors in V_t might be dominated by others, the optimal t-horizon value function can be represented by a parsimonious set V_t^-. The set V_t^- is the smallest subset of V_t that still represents the same value function V_t^*; all $\alpha-$vectors in V_t^- are useful at some belief state [10]. To compute V_{t+1} (and V_{t+1}^-), we only need to consider the parsimonious set V_t^-.

Though a lot of algorithms exist to compute V_{t+1}, the fastest of exact value-iteration algorithm can solve only the toy problems.

2.2 POMDP policy iteration

Value iteration takes a larger number of iteration to converge to infinite-horizon when the discount factor is large. Policy iteration finds the infinite-horizon policy directly and takes a smaller number of iterations over successively improved policies. The policy iteration algorithms iterate policies and try to improve the policies themselves. The iteration of policies $\pi_0, \pi_1, ..., \pi_t$ then converges to the optimal infinite horizon policy π^*, as $t \to \infty$. Policy iteration algorithms usually work in two phases, policy evaluation and policy improvement. In policy evaluation we compute the value function $V^{\pi}(b)$ and policy improvement improves the current policy π based on the value function of policy evaluation step.

Value iteration algorithms extract a policy from a value function, but policy iteration algorithms work in opposite direction. They first try to represent a policy so that its value function can be calculated. The first POMDP policy iteration algorithm was described in [15]. It used a cumbersome representation of a policy as a mapping from a finite number of polyhedral belief space regions to actions, and then converted it to a finite state controller (FSC) in order to calculate the policy value. The conversion between the two representations is extremely complicated and difficult to implement and policy iteration described in [15] is not used in practice.

2.3 The Q_{MDP} Value Method

Some other approaches seek learning using Q learning of the underlying MDP. Q learning is a reinforcement learning approach in MDP and assumes that probabilities or rewards are unknown. Q learning suggests defining a function Q, which corresponds to taking the action α_i in state ϕ_i and then continuing optimally or according to whatever policy one currently has.

$$Q(\phi_i, \alpha_i) = \sum_{\phi_j} P_{\alpha_i}(\phi_i, \phi_j)(R_{\alpha_i}(\alpha_i, \alpha_j) + \gamma V(\phi_j)) \tag{2.4}$$

While this function is also unknown, experience during learning is based on (ϕ_i, a) pairs (together with the outcome ϕ_j); that is, I was in state ϕ_i and I tried doing α_i and ϕ_j happened). Thus, one has an array Q and uses experience to update it directly. To solve the POMDP using Q function we temporarily ignore the observation model and find the $Q(\Phi, \alpha)$ values for the MDP consisting of transition and reward only. These values can be computed efficiently using dynamic programming approaches [8]. With Q values in hand, we can treat all the Q values for each action as a single linear function and estimate Q value for a belief state b in POMDP as

$$Q_\alpha(b) = \sum_{\phi} b(\phi) Q(\phi, \alpha) \tag{2.5}$$

This estimate amounts to assuming that any uncertainty in the agents current belief state will be gone after the next action.

The drawback of the policy is that it will not take action to gain information. For example a "look around without moving actions and a "stay in place and ignore everything" actions would be indistinguishable with regard to the performance of the policies under an assumption of one-step uncertainty. This can lead to situations in which the agent loops forever without changing belief state.

2.4 Replicated Q-Learning

[19] explores the problem of learning in POMDP model in a reinforcement-learning setting. The algorithm attempts to learn the transition and observation probabilities and uses an extension of Q-Learning [20] to learn approximate Q function for the learned POMDP Model.

Replicated Q-learning generalizes the Q-learning to apply to vector valued states and uses a single vector, q_α, to approximate the Q function for each action $\alpha : Q_\alpha(b) = q_\alpha.b$. The components of the vector are updated using

$$\Delta q_\alpha(\phi) = \beta\ b(\phi)(r + \gamma\ max_{\alpha\prime}\ Q_{\alpha\prime}(b\prime) - q_\alpha(\phi)) \tag{2.6}$$

The rule to update Q is evaluated for every $\phi \in \Phi$ and each time the agent makes a state transition. Here β is the learning rate, b the belief state, α the action taken, r the received reward in transition, and $b\prime$ the resulting belief state. The rule applies the Q-learning update rule to each component of q_α in proportion to the probability that the agent is currently occupying the state associated with that component. Simulating a series of transitions from belief state to belief state and applying the update rule at each step, this learning rule can be used to solve a POMDP. This rule reduces exactly to standard Q learning if observations of the POMDP are sufficient to ensure that agent is always certain of its state [18].

Though replicated Q-Learning is a generalization of Q learning, it does not work effectively to cases when the agent is faced with significant uncertainty. And since each component to predict Q values is adjusted independently, the learning rule tends to move all the components of q_α towards same value [18].

2.5 Linear Q-Learning

Similar to replicated Q-Learning is the Linear Q Learning algorithm. The difference being each component of q_α are adjusted to match the coefficient of the linear function that predicts the Q value rather than training each component of q_α towards the same value. This is done by applying the delta rule for neural network [21]. On adapting this rule to belief MDP framework it becomes as shown below:

$$\Delta q_\alpha(\phi) = \beta\ b(\phi)(r + \gamma\ max_{\alpha\prime} Q_{\alpha\prime}(b\prime) - q_\alpha.b) \tag{2.7}$$

Like the replicated Q-learning rule, this rule reduces to ordinary Q-Learning when the belief state is deterministic.

In neural network terminology training instance for the function $Q_\alpha(.)$ is the linear Q-learning view $(b, r + \gamma\ max_{\alpha\prime} Q_{\alpha\prime}(b\prime))$. While replicated Q-learning in contrast uses the same as training instance for the component $q_\alpha(\phi)$ for every $\phi \in \Phi$.

Linear Q-learning also has the same limitation as replicated Q-learning that it considers only linear approximation to the optimal Q functions.

We propose a different approach in which belief states of a POMDP are guessed using a tree structure. We call this a state estimation tree. We construct a tree that depends on the POMDP structure to estimate its states. Each node of the tree has a Learning Automata (LA). The depth of the tree can be changed depending on the model of POMDP. Each LA updates its actions when process comes back to the same belief state again, which means that the process comes back to the same node of the tree again.

3 STATE ESTIMATION

A POMDP is an MDP in which agent cannot observe the current state. Similar to an MDP, the goal of a POMDP is to maximize expected discounted future reward; however because of its insufficient knowledge regarding the current state, a POMDP makes an observation based on the action and resulting states.

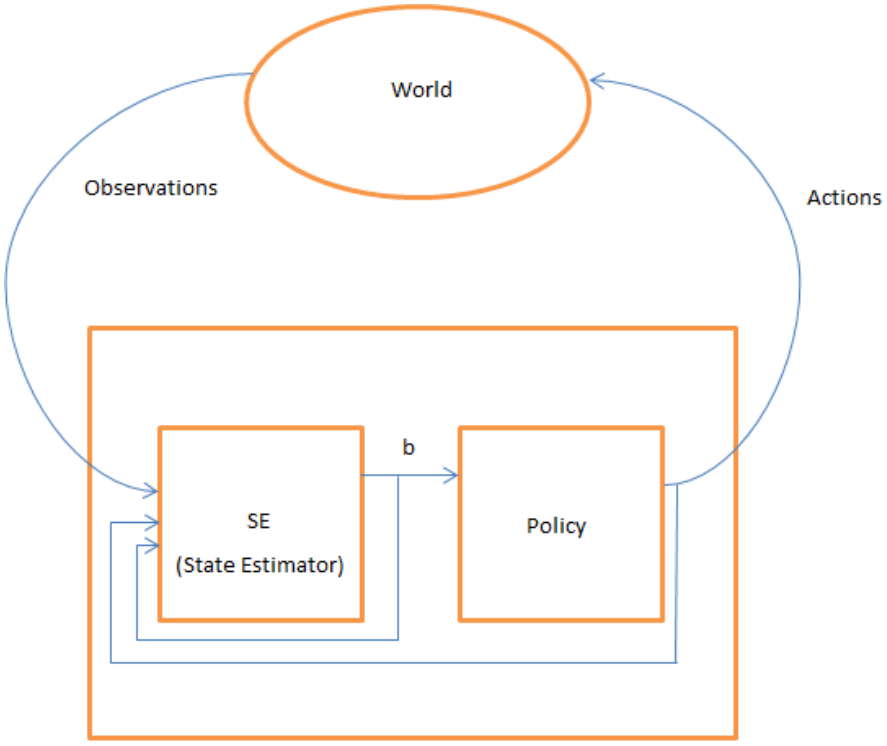

Figure 3.1. POMDP Agent decomposition

In Figure 3.1 of POMDP agent decomposition, the agent consist of two modules a state estimator and a policy. State estimator senses the observation from the outer

world and decides the current belief state b of the Agent. Based on this belief state b agent choses an action using the policy. These actions of the agent will have effect on the outer world of the agent and the agent will sense these changes in the outer world in form of observation and the process will go on.

The agent makes observation and generates actions. It keeps internal belief state b that summarizes its previous experience. The state estimator is responsible for updating the belief state based on the last action, current observation and previous action. Policy decides what action the agent should take given the agent's belief state. The belief state can be defined as the most probable state of the world, given the past experience. This might be a basis for action in some cases but it is not sufcient in general. In order to act effectively, an agent must take into account its own degree of uncertainty.
An agent can be categorized as following according to the internal states it maintains.

- A reactive(memoryless) agent defined by $\phi_t = Y_t$,
- One having a finite fixed length window of previous observationd $\phi_t = h_{t-k:t}$,
- A variable length markov model defined by a suffix tree ϕ_t,
- A recurrent neural network defined by ϕ_t,
- One where $\phi_t = P(X_t|h_1 : t)$ is a belief state (requires knowledge of environment model),
- A finite state machine defined by ϕ_t.

If the environment model of the POMDP is known, the optimal approach for the agent will be to compute sufficient statistic $b_t = P(X_t|h_{1:t})$, and use this as its internal state [22]. b_t is called the belief (information) state, and can be updated online using Bayes rule (sometimes called a state estimator).

As we have already discusses that a hidden Markov model (HMM) is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (hidden) states. The addition of actions and rewards at each state in a HMM gives rise to a POMDP. The Viterbi algorithm is a stochastic state estimator algorithm to solve HMM. This algorithm can be applied to simple scenarios like POMDPs to estimate the state of an agent based on the observation sequence. We will define a HMM and explain how it generates observation sequence and estimates state sequence. This discussion is based on [23]

A HMM can be described as following:

- A finite number of states of the environment, suppose n
  $\Phi = \phi_1, \phi_2, \phi_3, ..., \phi_n$
  Though the states are hidden and are not observed, for many practical applications there is often some physical significance attached to the states or set of states of the model. We denote a state at time t as q_t

- A finite number of distinct observation symbols per state, suppose m
  $O = o_1, o_2, o_3, ..., o_m$
  The observation symbols correspond to the physical output of the system being modeled

- The state transition probability distribution $P = [p_j^i]$ where

$$p_j^i = Prob[q_{t+1} = \phi_j | q_t = \phi_i]$$
$$1 \leq i, j \leq n, p_j^i \geq 0, p_j^i \leq 1, \sum_j p_j^i = 1$$

- The probability distribution of observation symbol in some state j, $B = b_j(k)$ where

$$b_j(k) = P[o_t = o_k | q_t = \phi_j]$$
$$1 \leq j \leq n, 1 \leq k \leq m$$

where $o(t)$ is the observed symbol at instance t

- The initial state distribution $\pi = \pi_i$ where

$$\pi_i = Prob[q_1 = \phi_i]$$

$$1 \leq i \leq n$$

If we observe a sequence $O = [o_1, o_2, o_1, , o_t]$ and are given a model of environment $\lambda = (P, B, \pi)$ we can use HMM to choose to estimate the optimal state sequence $Q = q_1, q_2, q_3, , q_t$.

Viterbi Algorithm is a formal technique that uses dynamic programming to discover one of the most likely state sequences for an observation. Following is the brief description of the algorithm.

Suppose we are given a HMM with state space Φ, initial probabilities p_i of being in state i and transition probability of transitioning from state i to state j is p_j^i. Given an observation sequence $O = [o_1, o_2, o_1, , o_t]$, we need to find the best state sequence $Q = [q_1, q_2, q_3, , q_t]$. The most probable sequence of hidden states is that combination that maximizes $Prob(observedsequence|hiddenstatecombination)$.

The approach to find most probable sequence of hidden states by finding the combination that maximizing $Prob(observedsequence|hiddenstatecombination)$ is viable, but to find it by exhaustively calculating each combination is computationally expensive. We can use the time invariance of the probabilities to reduce the complexity of the calculation [24]. We will consider recursively finding the most probable sequence of hidden states given an observation sequence and a HMM. We will first define the partial probability δ, which is the probability of reaching a particular intermediate state. δ represents the probability of the most probable path to a state at time t, and not a total.

For each intermediate and terminating state in the HMM, there is a most probable path to that state. These paths are called partial best paths. Each of these partial best paths has an associated probability, the partial probability or δ [25]. Thus $\delta(i,t)$ is the maximum probability of all sequences ending at state i at time t, and the partial best path is the sequence which achieves this maximal probability. Such a probability (and partial path) exists for each possible value of i and t. In particular, each state at time $t = T$ will have a partial probability and a partial best path [25]. We find the overall best path by choosing the state with the maximum partial probability and choosing its partial best path. In order to do that we need to define the quantity

$$\delta_t(i) = \max_{q_1,,q_t} \; Prob[q_1, ..., q_t = i, o_1, o_2, o_1, ..., o_t|\lambda] \tag{3.1}$$

Here $\delta_t(i)$ is the highest probability of ending in state ϕ_i while observing the observation sequence $[o_1, o_2, o_1, ..., o_t]$. By using the principle of induction we can also write it as

$$\delta_{t+1}(j) = \left[\max_i \delta_t(i) p_j^i\right] b_j(o_t) \tag{3.2}$$

To get the whole state sequence we need to keep track of the argument that maximizes Equation 3.2, for each t and j. we can do this using an array $\xi_t(j)$. The detailed description to find the best state sequence is as following:

INITIALIZATION: When t = 1 the most probable path to a state does not exist. We use the probability of being in that state given $t = 1$ and the observable state $o1$ as

$$\delta_t(i) = p_i b_i(o_1)$$
$$\xi_1(i) = 0$$

RECURSION:

$$\delta_t(j) = \max_{1 \le i \le n} \left[\delta_{t-1}(i) p_j^i\right] b_j(o_t)$$
$$2 \le t \le T$$

$$1 \leq j \leq n$$
$$\xi_t(i) = arg\ \max_{1 \leq i \leq n} \left[\delta_{t-1}(i) p_j^i\right]$$
$$2 \leq t \leq T$$
$$1 \leq j \leq n$$

TERMINATION:

$$p^* = \max_{1 \leq i \leq n} [\delta_T(i)]$$
$$q_T^* = \max_{1 \leq i \leq n} [\delta_T(i)]$$

PATH BACKTRACKING:

$$q_T^* = \xi_{t+1}(q_{t+1}^*)$$
$$t = t-1, t-2, ..., 1$$

Viterbi algorithm is a deterministic algorithm and provides a computationally efficient way of analyzing observations of HMMs to recapture the most likely underlying state sequence. But to use it we should have the complete knowledge of transition probabilities.

In our experiments we found that Viterbi algorithm maps observation sequence to states with 60% to 80% of accuracy but it requires complete knowledge about the model. After analyzing the Viterbi algorithm we propose a solution based on tree structure of the states of POMDP, which does not require prior knowledge of transition probabilities or rewards probability. The only information known is assumed to be the initial start state of the POMDP. This start state is called landmark state of the POMDP. We assume that agent has means to know when it comes back to landmark state.

Based on the number of states and actions available in the POMDP we create a tree structure. The root of the tree is the landmark state. Each node has child nodes

corresponding to actions available and observation symbols. Consider a POMDP that has n states in it. Each state in it can observe m discrete observations symbols. There are p actions available at each state. Then each node will have $m \times p$ child nodes, each corresponding to the action taken and observation symbol.

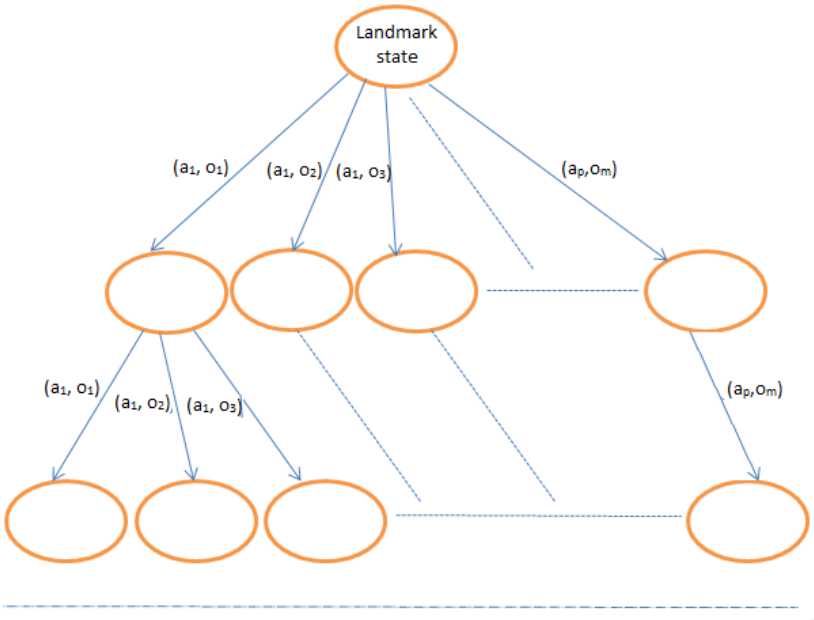

Figure 3.2. State Estimation

Landmark state is the only known state in our POMDP model. We assume no information is available about the transition probabilities and the reward matrix of the POMDP. The nodes in the tree correspond to the actual state in the POMDP. Each node may not correspond to a unique state and multiple nodes can correspond to a single state in POMDP. We don't have any information about what node corresponds to which state. Each node is mapped to some state in POMDP based on observation symbols observed and action taken at that state. The depth of the tree decides how many past observation we want to consider for the mapping of states. The state tree

tells us that at the beginning the agent is in the landmark state and he will move to other child nodes based on actions he chooses and the token he observe during the transition process.

When the POMDP starts transition we only observe the observation symbols and action taken by that state. Based on the actions taken and the observation symbols we correlate each node in the tree with some state in POMDP. For example we start from landmark state and take some action, suppose a_t, and observe a symbol, suppose o_t. This action may result in transition from current state to some other state. This next state will be mapped to a child node in our tree with link (a_t, o_t) from the current node.

The depth of the tree decides how many states we want to remember in tree form. Each node in the tree acts as a belief state of the POMDP. Each level of tree corresponds to information equivalent to 1 observation. After the number of observation exceeds the depth of the tree, we assume that POMDP is in some unknown state. We don't have any information about the state of the POMDP unless the landmark state is reached again. As we assume that the agent knows when it comes back to the landmark state, once we get back to the landmark state we start the same mapping process again.

We illustrate our state estimation approach in POMDP using following example. Consider a POMDP:

- Where number of states in POMDP are 4 so we have $\Phi = \{\phi_1, \phi_2, \phi_3, \phi_4\}$.
- For simplicity, we consider that there are only 2 actions available at each state, and they are same for all the stats. $\alpha = \{\alpha_1, \alpha_2\}$ for all the states.
- There are two discrete observations available per state $O = \{0, 1\}$. In this example we consider that these observations are the rewards that we get as a result of transition. So when we take an action and observe an observation

symbol 0 we get penalty (no reward), when we take an action and observe symbol 1 we get a reward.

- Agent transitions among the states with different transition probabilities for different actions. We don't have any information about these transition probabilities.
- In the beginning there is no knowledge present in a state so actions are chosen randomly in each state.

Following is diagrammatic representing our POMDP:

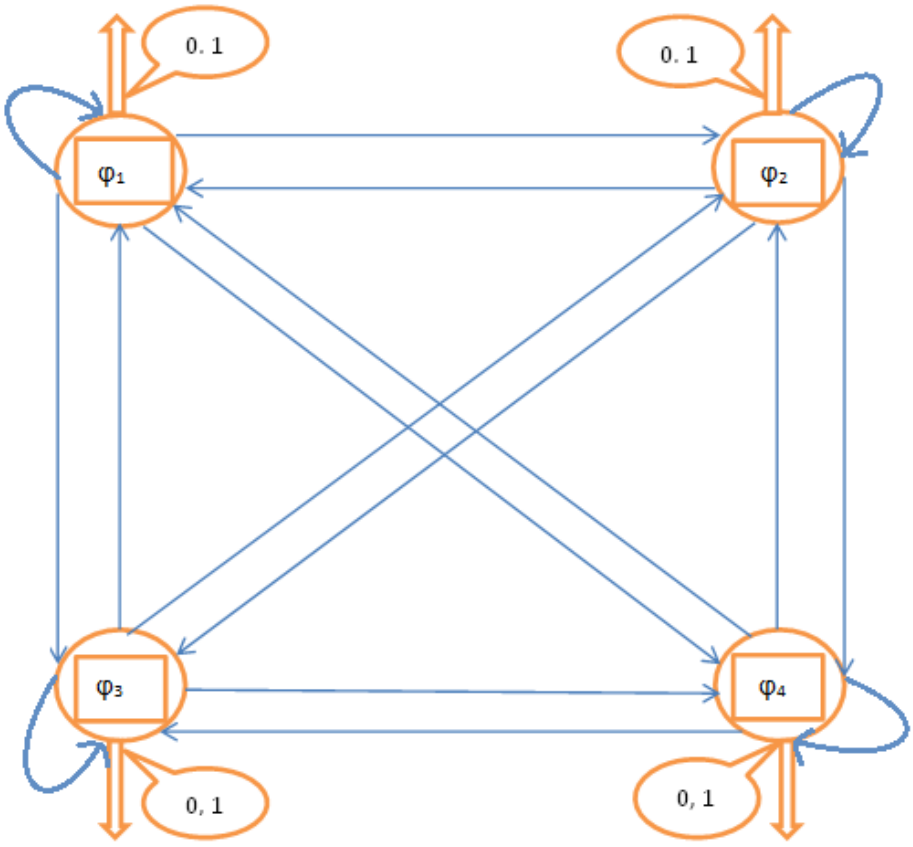

Figure 3.3. POMDP Example diagram

In Figure 3.3, an agent can transition among 4 states $(\phi_1, \phi_2, \phi_3, \phi_4)$. At each state we have two actions available (α_1, α_2). At each state we can observe one of the two symbols $(0, 1)$. We want to map the states of this POMDP with nodes in the state estimation tree. An agent at each state takes an action and emits an observation symbol, considered as reward. Our ultimate goal is to choose action at each state in such a way that long term reward is maximized.

To get the idea of current state of the agent we create the state estimation tree for this POMDP. Consider a tree with depth 3. A tree with depth 3 implies that we can store observation symbols up to 3 observations and map states to tree nodes for 3 continuous observations starting from root node (Landmark State). The state estimation tree will look like following:

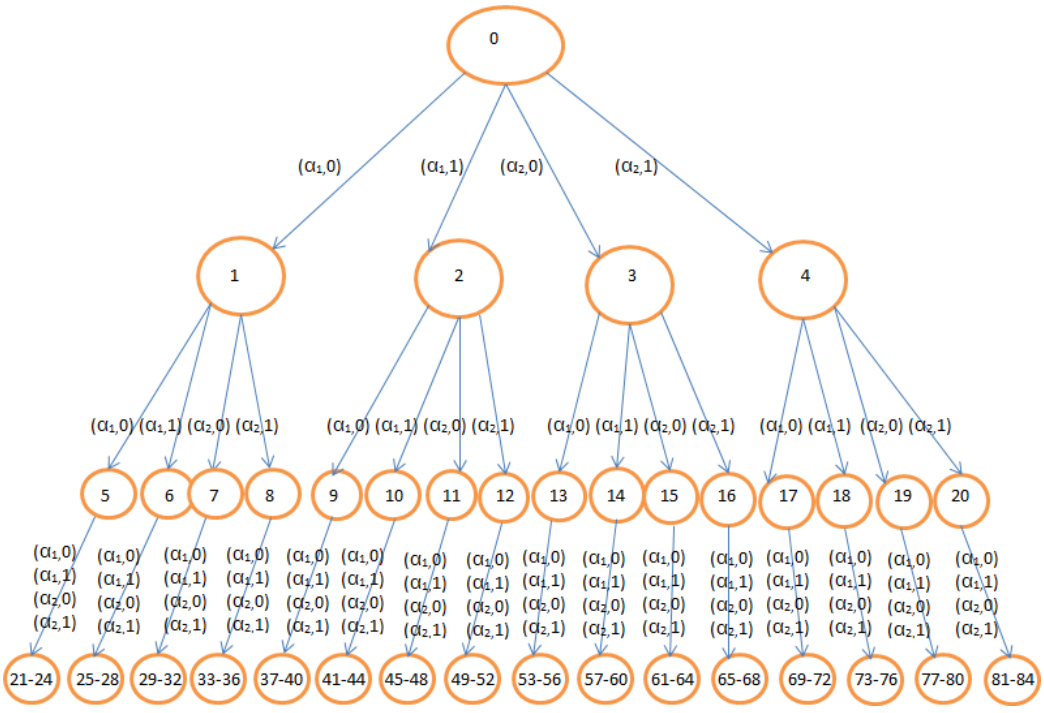

Figure 3.4. State Estimation Example

In the state estimation tree, root node 0 is the landmark state. Suppose this landmark state is state ϕ_1. Whenever the agent is in this state, it knows that the state is ϕ_1. The depth of the tree is 3. It can map tree nodes with the states only up to 3 continues observations. At the landmark state if the action α_1 is taken and the observation token 0 is received the current state of the agent is node 1. At node 1 if action α_2 is taken and observation token 1 is received the current state will become node 8. Once we get to the leaf nodes, we are out of tree nodes to map to the states, so we assume that the agent is in some state that we don't know about. As soon as the agent gets back to the landmark state ϕ_1 at any time during transition in unknown states the mapping process is started again starting from the root node 0.

Empirical analysis was done on POMDP states and nodes in the state estimation tree. POMDP transition probabilities were chosen randomly and it was found that 60% to 90% of the times same nodes are mapped to same POMDP state. For the POMDP example of Figure 3.3, with random transition probabilities and state estimation tree of depth 3 as in Figure 3.4, we observed the following mapping between states of the POMDP and Nodes of the state estimation tree.

$\phi_1 \rightarrow 0$	$\phi_1 \rightarrow 4$
$\phi_3 \rightarrow 19$	$\phi_4 \rightarrow 80$
$\phi_3 \rightarrow \mathrm{R}$	$\phi_4 \rightarrow \mathrm{R}$
$\phi_1 \rightarrow 0$	$\phi_4 \rightarrow 1$
$\phi_3 \rightarrow 7$	$\phi_1 \rightarrow 29$
$\phi_4 \rightarrow \mathrm{R}$	
$\phi_1 \rightarrow 0$	$\phi_1 \rightarrow 4$
$\phi_2 \rightarrow 17$	$\phi_3 \rightarrow 72$
$\phi_4 \rightarrow \mathrm{R}$	
$\phi_1 \rightarrow 0$	$\phi_2 \rightarrow 4$
$\phi_3 \rightarrow 20$	$\phi_1 \rightarrow 82$

$\phi_3 \rightarrow$ R

$\phi_1 \rightarrow 0$ $\phi_4 \rightarrow 1$

$\phi_3 \rightarrow 7$ $\phi_4 \rightarrow 32$

$\phi_1 \rightarrow 0$ $\phi_4 \rightarrow 1$

$\phi_3 \rightarrow 7$ $\phi_4 \rightarrow 32$

$\phi_2 \rightarrow$ R

R denotes when we are out of the state estimation tree and cannot map states of the POMDP with state estimation tree. Observation sequence of length 30 was considered for this example. Summarizing the state estimation we get that

$\phi_1 \rightarrow$ 0(6), 4(2), 29(1), 82(1)

$\phi_2 \rightarrow$ 4(1), 17(1)

$\phi_3 \rightarrow$ 7(3), 19(1), 20(1), 72(1)

$\phi_4 \rightarrow$ 4(3), 80(1), 32(2)

In this mapping the only ambiguity we see is with Node 4, which is mapped to state ϕ_1 twice and with state ϕ_2 once.

4 LEARNING IN POMDP USING TREE

Learning problem in POMDP when the knowledge of transition probabilities and rewards is present is common and has been studies by many researchers. This learning problem can be stated as follows. Let $\phi = \{\phi_1, \phi_2, ..., \phi_n\}$ be the state space of a finite POMDP. And $\alpha^i = \{\alpha^i_1, \alpha^i_2, ..., \alpha^i_{r_i}\}$ be the finite set of actions/decisions available in each state ϕ_i. The transition probabilities $t^i_j(k)$ and rewards $r^i_j(k)$ depend on the starting state ϕ_i, the ending state ϕ_j, and the action $\alpha^i_k(k = 1, ..., r_i)$ used in ϕ_i. The goal is to choose the set of actions, or policy, $\alpha = \{\alpha^1_{i_1}, \alpha^2_{i_2}, ..., \alpha^N_{i_N}\}$ that maximizes

$$J(\alpha) \triangleq \Delta \frac{1}{n} E\left[\sum_{t=0}^{n-1} r(x(t), x(t+1), \alpha)\right] \tag{4.1}$$

where $r(x(t), x(t+1), \alpha)$ is the reward generated by a transition from $x(t)$ to $x(t+1)$ using the policy α. The set of policies is limited in this formulation to stationary nonrandomized policies. Hence, the best strategy in any state is a pure strategy and is independent of the time at which the state is occupied. Under the following assumption, it can be shown that the optimal α belongs to this set of policies [11].

Here we make the following assumption:
ASSUMPTION: The Markov chain corresponding to each policy α is Ergodic.
Therefore we can say that there are no transient states and a limiting distribution $\pi(\alpha) = (\pi_1(\alpha), \pi_2(\alpha), ..., \pi_N(\alpha))$ exists, with each $\pi_i(\alpha) > 0$, which is independent of the initial state [17]. According to [17] under this assumption, the expected reward per step $J(\alpha)$, defined in Equation 4.1, can also be written in terms of the limiting (stationary) probabilities $\pi(\alpha)$ as

$$J(\alpha) = \sum_{i=1}^{N} \pi_i(\alpha) \sum_{j=1}^{N} t^i_j(\alpha) r^i_j(\alpha) \tag{4.2}$$

Learning models have been studied extensively by psychologists and systems theorists from both modeling and control viewpoints [26], [27], [28]. A particularly simple model for sequential decision making in unknown random environments is the learning automaton [29]. There are many variation of learning automaton proposed but the basic idea remains the same. The automaton has a finite set of actions and based on environment response, it updates a probability distribution over the actions. Using this approach, the automaton tries to learn from the environment [17].

Let $\alpha = \alpha_1, \alpha_2, ..., \alpha_r$ be the automaton actions available in a state and β be the environment response set. Each element of β is normalized to lie in the interval $[0, 1]$. Here 1 is the best response and 0 is the worst response. Given that at time n the action vector is $\alpha(n) \in \alpha$, the response is $\beta(n) \in \beta$ and actions probabilitiy vector is $p(n) = p_1(n), p_2(n), ..., p_r(n)$. Then the way the action probabilities over vector $p(n)$ are updated is decided by the learning algorithm T of automaton so $p(n+1) = T[p(n), \alpha(n), \beta(n)]$.

The essential properties of any automaton can be exhibited by the following linear algorithm with various parameter values [17].
If $\alpha(n) = \alpha_i$, then

$$p_i(n+1) = p_i(n) + a\beta(n)[1 - p_i(n)] - b[1 - \beta(n)]p_i(n) \tag{4.3}$$

$$p_j(n+1) = p_j(n) - a\beta(n)p_j(n) + b(1 - \beta(n)) \left[\frac{1}{r-1} - p_j(n)\right] \tag{4.4}$$

$$0 < a < 1 \; and \; 0 \leq b < 1$$

a and b are called reward and penalty parameters respectively. We assume that all initial probabilities $p_i(0)$ lie in the interval $(0, 1)$. The learning scheme depends on the value of a and b. if $a = b$ then the learning scheme is linear reward-penalty. If $b < a$ then it is called linear reward-ϵ-penalty and if $b = 0$ then it is Linear reward-inaction L_{R-I} [17]. Though any of the schemes can be used, we consider (L_{R-I}) in this thesis.

Convergence of (L_{R-I}) can be demonstrated by the observation that it makes a Markov process with stationary transition probabilities. Given that environment response $\beta(n)$ for the actions $\alpha(i)$ are independent of time, the probability of $p(n+1)$ is determind by $p(n)$. We assume that $p(n)$, defined in 4.4, is a random vector.

If for any n unit simplex S_r is defined as

$$S_r \triangleq \Delta \left\{ p | 0 \leq p_i \leq 1, \sum_{i=1}^{r} p_i = 1 \right\} \tag{4.5}$$

Equation 4.4 is linear and has an interesting distance diminishing property for a and b. This property makes Markov process compact. The mapping T defined on S_r is distance diminishing on S_r (T is a stochastic contraction) if, for any pair of points p^1 and p^2 belonging to S_r [17]

$$\sup_{p^1 \neq p^2} \frac{d(T[p^1, \alpha, \beta], T[p^2, \alpha, \beta])}{d(p^1, p^2)} < 1 \tag{4.6}$$

where $d(x^1, x^2)$ is the Euclidian distance matrix. The convergence of the $L_R - I$ scheme can be proved by asymptotic behavior of compact Markov processes [17].

If $\beta(n)$ is a measure of relative success where $\beta(n) = 1$ equals to the maximum success, then the expected success can be defined as

$$M(n) \triangleq \Delta \; E[\beta(n)|p(n)] = \sum_{i=1}^{r} E[\beta(n)|\alpha_i] p_i(n) \tag{4.7}$$

In [27], $\max_i E[\beta(n)|\alpha_i]$ has been defined as d_0 and the following theorem has been proved:

Theorem: For any $\epsilon > 0$ and any $p(0)$ in the open simplex S_r, there exists an $0 < a^* < 1$ such that for $b = 0$ and any $a < a^*$ in Equation 4.4

$$\lim_{n \in \infty} E[M(n)] > d_0 - \epsilon \tag{4.8}$$

In [27] the proof has been given for the case when $\beta(n)$ is a binary random variable. That means $\beta = 0, 1$. [30] proves the result termed as $\epsilon - optimality$ for general $\beta(n)$.

4.1 Automata Games and Decision Making in POMDP

This POMDP control is not a problem when the automatons exist as individual, but it becomes a principal issue when many automatons are interconnected. In [31] connection between decentralization of interconnected automaton and abstract games was unveiled.

An automaton game involves N automaton $Ai(i = 1, 2, ..., N)$. Each of the automaton has an action set $\alpha^i = \{\alpha_1^i, \alpha_2^i, ..., \alpha_{r_i}^i\}$. The automatons interact through a stationary random environment. At each instant n, an automaton A_i selects an action according to its current probability distribution $p^i(n) = (p_1^i, p_1^i, ..., p_{r_i}^i)$. The cumulative action $\alpha(n) = \alpha = \alpha_{i_1}^1, \alpha_{i_2}^2, ..., \alpha_{i_N}^N$ which we choose depending on the probability $p(\alpha) = \Pi_{j=1}^N p_{i_j}^j(n)$, determines the distribution of received random response $\beta^i(n)$. A stationary random environment means that the response distribution is fixed over time and each automaton has access only to its own response. In this game formulation, no player is aware of other players, other player"s action at any instance of time or action response distribution.

In the POMDP problem also the automaton at each state will probabilistically choose an action and based on the reward will get a response and update its action probabilities. The only information available to the automaton is the reward received and it depends upon the actions taken by other automaton when the chain is in other states. So automata game and decentralized control of Markov chain are very similar in this sense.

4.2 Learning as a Control Strategy for POMDP

We have discussed creation of a state estimation tree in the previous section. The learning algorithm proposed here is based on the state estimation tree. Each node of the state estimation tree will have an automaton in it. Automaton in each node is

unaware of the other automatons available in other nodes of the tree. The algorithm updates the action probability only when the process returns to the same node again. The only known state in the tree is the root node of the tree considered as landmark state. The tree knows its state when it gets back to the landmark state.

4.2.1 The automaton updating procedure

We propose the learning approach that involves one learning automaton for each action state. So each of the nodes of the state estimation tree has an automaton (A_i) in it. A coordinator is present to perform the simple administrative tasks. Each automaton works on its local time scale $n_i = 0, 1, 2, ...$ The Markov chain corresponding to the POMDP operates on the global time scal $n = \{0, 1, 2, ...\}$. The coordinator guesses the current node in the state estimation tree for the POMDP based on past observations. The coordinator then activates the automaton A_i present at that node. The automaton A_i chooses an action α_k^i based on current action probability vector $p^i(n_i)$. If the POMDP is in an unknown state, that is the observation has passed the height of the tree, then a random action is chosen.

An important feature of the control scheme is that A_i does not get reward information from the current action. So the current one step reward $r_j^i(k)$ is unknown to it. A_i does not receive any information about the effect of its current action or about the activity of the Markov process. It gets this information only at time $n_i + 1$ when the control returns to the same node of the state estimation tree. At that time the automaton A_i at that node receives two pieces of information from the coordinator [17]:

- The cumulative reward generated by the process up to time n,
- The current global time n.

Based on this information the automaton A_i computes the incremental reward $\Delta\rho_k^i(n_i)$ generated since last local time $n_i(\alpha^i(n_i) = \alpha_k^i)$ and the corresponding elapsed global

time $\Delta\eta_k^i(n_i)$. The increments $\Delta\rho_k^i(n_i)$ and $\Delta\eta_k^i(n_i)$ are added to the current cumulative totals $\rho_k^i(n_i)$ and $\eta_k^i(n_i)$. This addition results in new cumulative totals $\rho_k^i(n_i+1)$ and $\eta_k^i(n_i+1)$. The environment response is then calculated as:

$$\beta^i(n_i+1) = \frac{\rho_k^i(n_i+1)}{\eta_k^i(n_i+1)} \tag{4.9}$$

As we have already discussed that the reward $r_j^i(k)$ is normalized in the interval $[0,1]$ so $\beta^i(n_i)$ and as a result $\beta^i(n_i+1)$ also lie in $[0,1]$.

When we come to the leaf nodes of the maintained tree, we run out of automaton and start choosing actions randomly (with equal probabilities). We continue this until we come back to the known, "landmark" state, in which case we start choosing actions again using the appropriate automata in the tree, starting with the root automata. This process will continue forever.

When we come back to the known, landmark state, we update the root automaton using the turn-around cumulative reward and updating scheme defined in Equation 4.4. The environment response for updating the automaton is defined by Equation 4.9. Similarly, we keep on updating the automaton used along the path followed in the state estimation tree, in exactly similar manner.

Each A_i is assumed to use the updating scheme defined in Equation 4.4 with $\beta(n)$ defined by Equation 4.9.This modified scheme is denoted by $T1$ and can be briefly described as following:

- At time n when coordinator chooses an automaton A_i, only A_i updates its action probabilities.
- When we come to the leaf nodes of the maintained tree and run out of automata we start choosing actions randomly but the reward generated and one instant of global time are added to their respective current totals by the coordinator.

- In the intervening time between two visits to any node of the tree, no knowledge of the sequences of states visited is provided to A_i. Only the current values of total reward and n are needed and these are incremented at each n whether any automaton in the tree is active.

All the work of the coordinator is of book keeper and not a decision maker. All the decision and estimations are performed by the automata and not by the coordinator.

4.2.2 Ergodic finite Markov chain property

Consider an N state Markov chain which has N_a action states $\phi_i^* \in \phi^*$, with action set $\alpha^i = \{\alpha_1^i, \alpha_2^i, ..., \alpha_{r_i}^i\}, r_i \geq 2$ in each state. If we associate one decision maker A_i with each ϕ_i^* then $T = \{N_a, \vartheta, J\}$ denotes a finite identical payoff game among $\{A_i\}$ in which the play $\alpha \in \vartheta = \alpha^1 \otimes \alpha^2 ... \otimes \alpha^{N_a}$ results in the payoff $J(\alpha)$ and can be defined as

$$J(\alpha) = \sum_{i=1}^{N} \pi_i(\alpha) \sum_{j=1}^{N} t_j^i(\alpha) r_j^i(\alpha)$$

and it is same as Equation 4.2. This finite identical payoff game T has a unique equilibrium [17].

Proof: Assume that a dummy decision maker with only one action is also associated with each non-action state. In the corresponding $N-$player game $T\prime$, a play (policy) has N rather than N_a components. However, since the action sets in $N - N_a$ states are degenerate, any play α in T is equivalent to a play in $T\prime$ in which the N_a action state decision maker use α. The theorem for $T\prime$ has also been proved in [17].

The proof for $T\prime$ is related to the convergence of policy iteration method discussed in [9]. Assume that a play α is a non-optimal equilibrium point (EP) of $T\prime$. According to [9], the gain $J(\alpha)$ and the relative values $v_i(\alpha)$ associated with α are found as the solution to

$$J(\alpha) = q^i(\alpha) + \sum_{j=1}^{N} t_j^i(\alpha) v^j(\alpha) - v^i(\alpha) \tag{4.10}$$

$$\text{Where } i = 1, 2, ..., N$$
$$\text{And } q^i(\alpha) = \textstyle\sum_{j=1}^{N} t_j^i(\alpha) r_j^i(\alpha)$$

$v^N(\alpha)$ is set to 0 arbitrarily to guarantee a unique solution. We can find a better policy than α using policy iteration. Suppose that state i is one of the states in which the better play differs from α. Consider the play β which differs from α only in state i. The component of β used in state i is found as that k which maximizes the following "test quantity"

$$\tau_i(k, \alpha) = q^i(k) + \sum_{j=1}^{N} t_j^i(k) v^j(\alpha) - v^i(\alpha) \tag{4.11}$$

$$\text{Where } k = 1, 2, ..., r_i$$

The test quantity depends on α since the values $v^j(\alpha)$ in the Equation 4.11 are kept as the ones computed from the original play α. A useful property of the test quantities is that for any play β, $J(\beta) = \sum_{j=1}^{N} \pi_j(\beta)\tau_j(\beta, \alpha)$, where $\pi_j(\beta)$ is the steady state probability for state j under play β and $\tau_j(\beta, \alpha)$ is the test quantity formed in state j by using play β in state j but relative values associated with play α [9]. Assumption about the Markov chain that Markov chain corresponding to each policy α is ergodic assures the existence of $\pi_j(\beta)$ for all j and β.

If we maximize Equation 4.11 then

$$\tau_i(\beta, \alpha) = \tau_i(\alpha, \alpha)$$
$$\tau_j(\beta, \alpha) = \tau_j(\alpha, \alpha)$$

From Equations 4.10 and 4.11 we know that $\tau_j(\alpha, \alpha) = J(\alpha)(j = 1, 2, ..., N)$. This follows:

$$J(\beta) = \sum_{j=1}^{N} \pi_j(\beta)\tau_j(\beta, \alpha) > \sum_{j=1}^{N} \pi_j(\beta)\tau_j(\alpha, \alpha)$$

Since β is superior to α and differs from α in one state α cannot be an equilibrium point and optimal policy in $T\prime$ is the optimal policy.

The uniqueness of an equilibrium in T is an interesting property. In any controlled Markov chain satisfying that each policy α is ergodic, the second best policy differs from the optimal policy only in the action chosen in exactly one state. In general, the k^{th} best policy can differ from the optimal policy in at most the actions chosen in $k - 1$ states [17].

4.2.3 Convergence

In the previous section about property of ergodic Markov chain, it was discussed that a Markov chain represented as an N automata game has a unique equilibrium. A payoff in automata game T is obtained by using a fixed policy. Each decision maker uses a updating procedure $T1$. T can be viewed as a limiting game $T = \lim_{n\to\infty} T(n)$. The elements of $T(n), s(\alpha,\ n) \triangleq \Delta E[\beta(n)|\alpha(n) = \alpha]$ depends on n. So the automata updating in POMDP is not the same as in an automata game. But from the assumption of ergodic Markov chain it can be deduced that $lim_{n\to\infty} s(\alpha,\ n) = J(\alpha)$ [17]. Here $J(\alpha)$ is the same as defined in ergodic Markov chain property.

$$J(\alpha) = \sum_{i=1}^{N} \pi_i(\alpha) \sum_{j=1}^{N} t_j^i(\alpha) r_j^i(\alpha)$$

Given that we can say that for a large n the ordering among $J(\alpha)$ will be similar to the ordering among $s(\alpha,\ n)$. Thus we can analyze the automata game T for the POMDP problem. If all the states in our state estimation tree are mapped correctly then the problem reduces to the control of Markov processes and the policy we get by out learning method will be $\epsilon - optimal$. However due to the uncertainty of state

in POMDP, the states in state estimation tree cannot be estimated accurately. Once the process reaches to the leaf nodes, we start taking actions randomly so the policy we defined is not the optimal policy but it will be $k-optimal$. Where k is the number of states that are not estimated properly. The policy described for POMDP will be $k-optimal$ as we have already discusses in previous section that k^{th} best policy in the Markov chain differs from the optimal policy in at most the actions chosen in $k-1$ states.

5 RESULTS

We run our proposed learning algorithm on various POMDP problem having 4 to 6 states. The results for long term reward were compared with completely random learning approach and with the underlying MDP having exact knowledge [17]. We performed this analysis with state estimation tree of different state estimation tree heights and with different transitions and reward probabilities of underlying MDP. Transition probabilities and reward probabilities were randomly generated in the experiments. Long term rewards are normalized in range between $(0, 1)$. The results have been analyzed on varying number of iteration for all the learning approaches. In random learning approach there is no learning and each state takes action randomly based on the initial probability distribution.

The results show that long term rewards with the proposed learning approach are always better than random action strategy but is less than optimal policy reward. The optimal policy rewards are calculated using the underlying MDP of our POMDP problem. When we have a complete knowledge of the states, POMDP turns into an MDP; and because of the complete knowledge of the states, optimal policy outperforms our approach. The results of proposed learning approach depend on the depth of the state estimation tree and with increase in the depth of the state estimation tree, the long term rewards are improved.

Experiments were run on the server machine with Xeon 5335 quad-core processors and 8GB of RAM. The time taken in learning is mentioned next to the learning methods in the following result graphs.

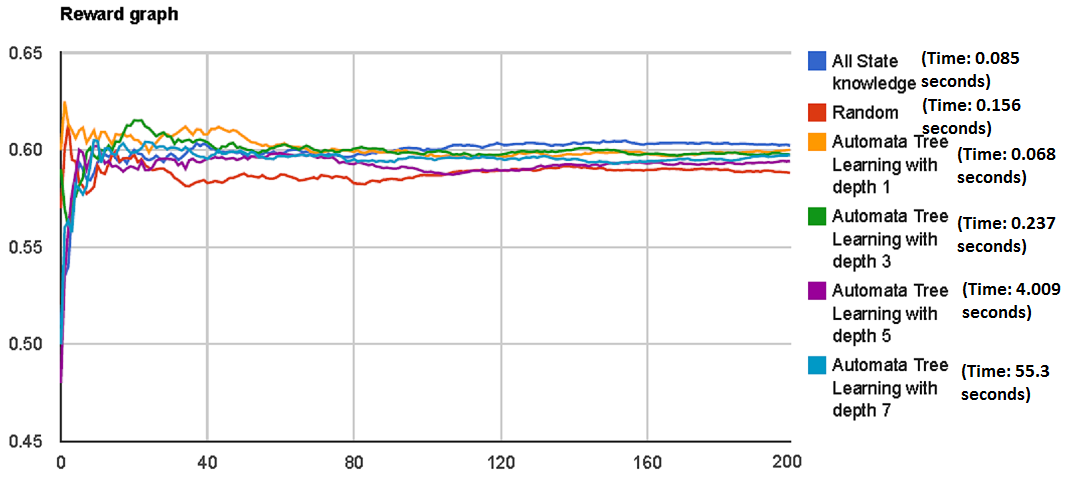

Figure 5.1. Normalized long term reward in a POMDP with 6 states over 200 iterations

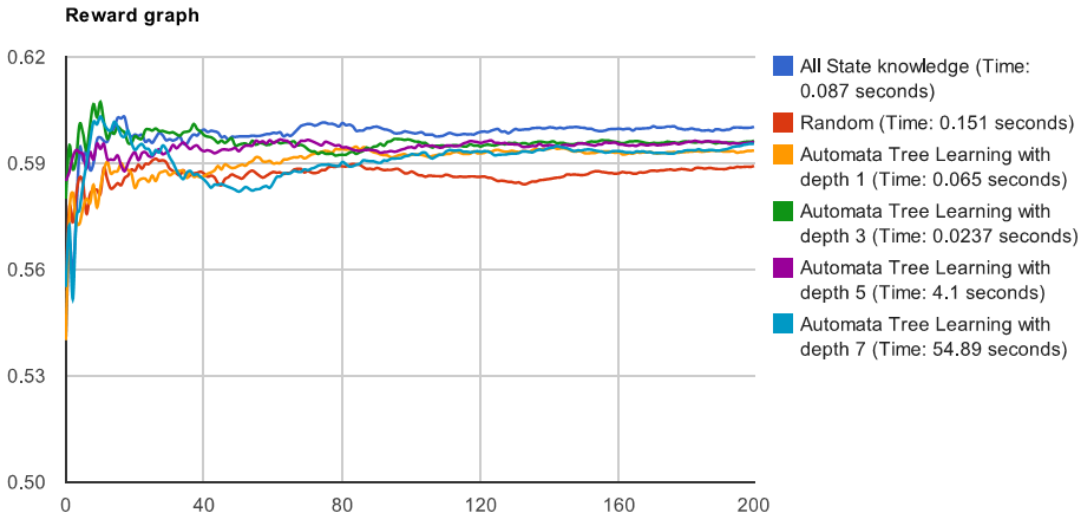

Figure 5.2. Normalized long term reward in a POMDP with 4 states over 200 iterations

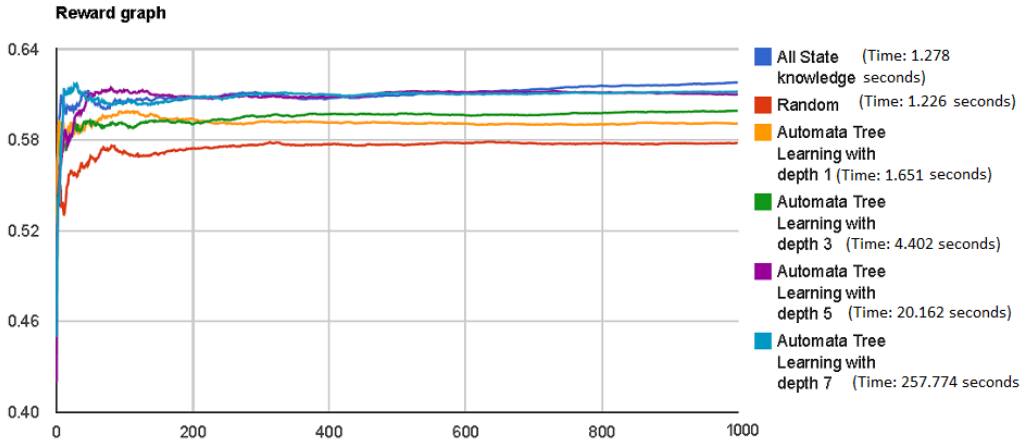

Figure 5.3. Normalized long term reward in a POMDP with 4 states over 1000 iterations

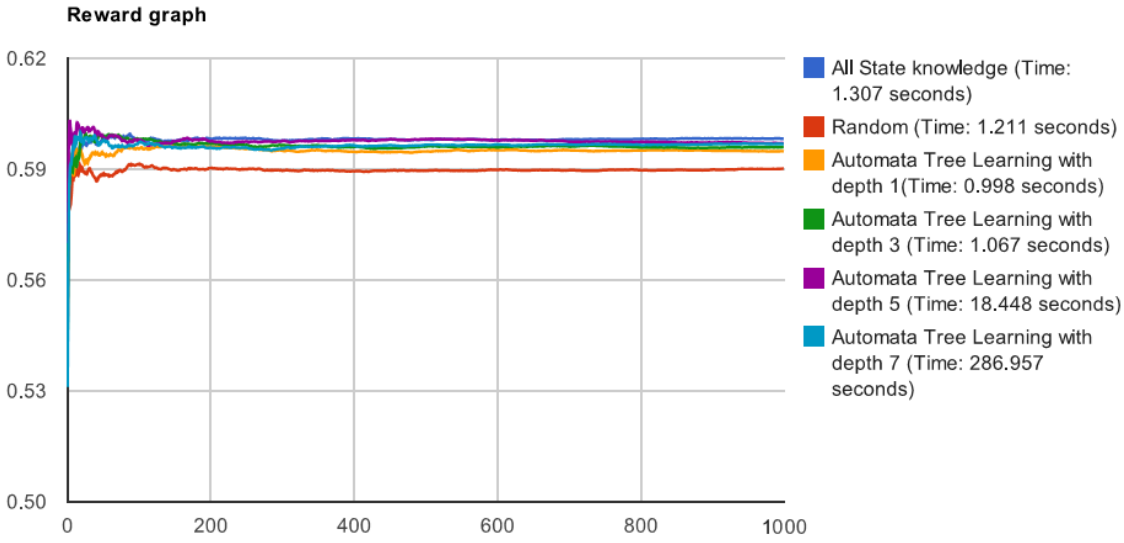

Figure 5.4. Normalized long term reward in a POMDP with 6 states over 1000 iterations

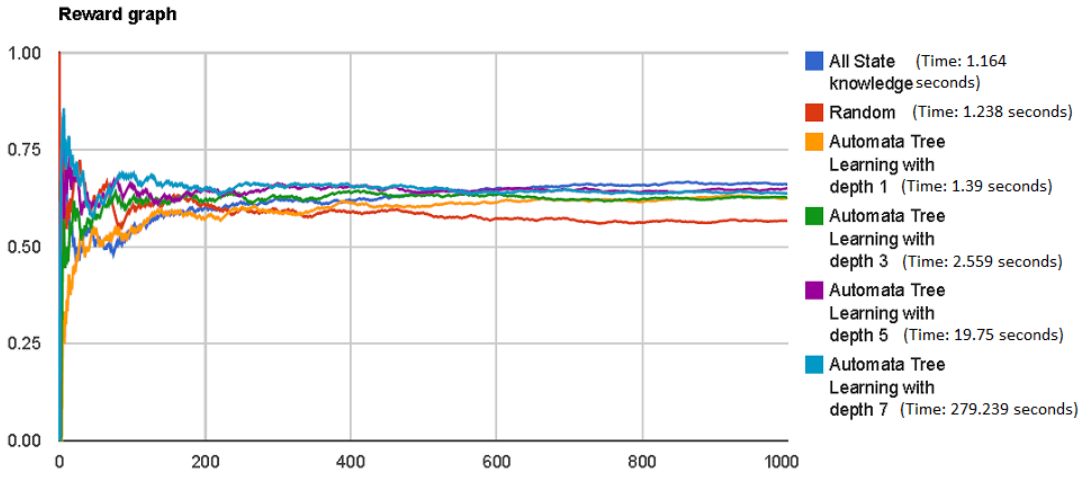

Figure 5.5. Normalized long term reward in a POMDP with 6 states over 1000 iterations

In result diagrams we see that the long term rewards in POMDP with automata tree learning lies in between random learning reward and optimal reward. In random transition learning approach the agent at each state takes random action and does not learn any information to maximize the rewards. The all state knowledge learning approach has complete knowledge about its current state, hence does not need to guess the current state. The all state knowledge learning approach gives the optimal reward as there is no uncertainty involved about the current state. In the diagrams we see that as the tree height increases the long term average reward tends towards the optimal reward. Along with the depth the learning time in the POMDP also increases. There is no significant gain in the reward of learning with automata tree with depth 5 and with depth 7; so we can stop increasing the depth of the learning tree when there is no significant gain of reward.

6 CONCLUSION AND FUTURE WORK

The basic problems addressed in this thesis are learning in POMDP and adaptation with minimum prior knowledge. The proposed solution does not require prior knowledge of transition probabilities and rewards. The proposed approach can also adapt to the changes in the environment. The approach presented in the thesis uniquely combines flexibility of learning theory with the structure of underlying MDP for the given POMDP. Main observation about the proposed learning approach can be summarized as:

- The proposed learning approach does not require any prior knowledge of the model of POMDP.
- The long term reward that we get is not optimal as states of the POMDP cannot be estimated with full accuracy. But the rewards are $k - optimal$ and are between random rewards and optimal rewards.
- Increasing the height of state estimation tree (ie knowledge about past observations) improves the long term rewards.

The results summarize that even without any prior knowledge of the POMDP; suggested learning method gives a policy for the POMDP that performs better than the random transitions. Otherwise the best policy without any prior information will be a random policy.

Future work can be done to estimate states more accurately. The current approach takes random action once we pass the leaf nodes of state estimation tree until we get back to the landmark state again and reinitialize the process. If numbers of states are very large then it may take infinite amount of time to get back to the landmark state.

And in that case the performance of the suggested learning approach will deteriorate. We can further explore cases when probability of getting into landmark state is fairly less. Another open issue is as the depth of the state estimation tree increases the number of false state estimation will also increase. Further analysis can be done on ideal depth of the state estimation tree for a given POMDP with N number of states.

LIST OF REFERENCES

LIST OF REFERENCES

[1] Wikipedia. Markov chain. `http://en.wikipedia.org/wiki/Markov_chain`.

[2] M. Mohamed and P. Gader. Handwritten word recognition using segmentation-free hidden Markov modeling and segmentation-based dynamic programming techniques. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 18(5):548 –554, may 1996.

[3] Thad Starner and Alex Pentland. Real-time american sign language recognition from video using hidden Markov models, 1996.

[4] Julian Kupiec. Robust part-of-speech tagging using a hidden Markov model. *Computer Speech and Language*, 6(3):225 – 242, 1992.

[5] Bryan Pardo and William Birmingham. Modeling form for on-line following of musical performances. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, pages 9–13. AAAI, 2005.

[6] L. Satish and B.I. Gururaj. Use of hidden markov models for partial discharge pattern classification. *Electrical Insulation, IEEE Transactions on*, 28(2):172 –182, apr 1993.

[7] E.A. Feinberg and A. Shwartz, editors. *Handbook of Markov Decision Processes - Methods and Applications*. Kluwer International Series, 2002.

[8] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

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

[10] Darius Braziunas. Pomdp solution methods. Technical report, 2003.

[11] S. M. Ross. *Applied Probability Models with Optimization Applications*. Holden-Day, San Francisco, 1970.

[12] P. Mandl. Estimation and control in Markov chains. *Advances in Applied Probability*, 6(1), 1974.

[13] V. Borkar and P. Varaiya. Adaptive control of Markov chains, i: Finite parameter set. *Automatic Control, IEEE Transactions on*, 24(6):953 – 957, dec 1979.

[14] Christos Papadimitriou and John N. Tsitsiklis. The complexity of markov decision processes. *Math. Oper. Res.*, 12(3):441–450, August 1987.

[15] E J Sondik. The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Operations Research*, 26(2):282–304, 1978.

[16] Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman. Acting optimally in partially observable stochastic domains, 1994.

[17] Richard M. Wheeler and Kumpati S. Narendra. Decentralized learning in finite markov chains. In *Decision and Control, 1985 24th IEEE Conference on*, volume 24, pages 1868 –1873, dec. 1985.

[18] Michael L. Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 362–370. Morgan Kaufmann, 1995.

[19] Lonnie Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *In Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 183–188. AAAI Press, 1992.

[20] C. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge,England, 1989.

[21] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart, J. L. McClelland, et al., editors, *Parallel Distributed Processing: Volume 1: Foundations*, pages 318–362. MIT Press, Cambridge, 1987.

[22] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *ARTIFICIAL INTELLIGENCE*, 101:99–134, 1998.

[23] Lawrence R. Rabiner. Readings in speech recognition. chapter A tutorial on hidden Markov models and selected applications in speech recognition, pages 267–296. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[24] Roger David Boyle. Hidden markov models. `http://www.comp.leeds.ac.uk/roger/HiddenMarkovModels/html_dev/main.html`.

[25] Kai Jiang. Viterbi Algorithm. `http://www.kaij.org/blog/?p=113`.

[26] Robert R. Bush and Frederick Mosteller. *Stochastic Models for Learning*. New York: Wiley, 1958.

[27] M. Frank Norman. Academic press, new york, 1972.

[28] M. L. Tsetlin. *Automaton Theory and Modeling of Biological Systems*. New York: Academic, 1973.

[29] K. Narendra and M. Thatcher. Learning automata: a survey. Levine's working paper archive, David K. Levine, 2010.

[30] S. Lakshmivarahan and M. A. L. Thathachar. Absolute expediency of q-and s-model learning algorithms. *Systems, Man and Cybernetics, IEEE Transactions on*, SMC-6(3):222 –226, march 1976.

[31] Richard M. Wheeler and Kumpati S. Narendra. Learning models for decentralized decision making. In *American Control Conference, 1985*, pages 1090 –1095, june 1985.