Hidden-Mode Markov Decision Processes for Nonstationary Sequential Decision Making

Samuel P. M. Choi, Dit-Yan Yeung, and Nevin L. Zhang

Department of Computer Science,
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
{pmchoi,dyyeung,lzhang}@cs.ust.hk

1 Introduction

Problem formulation is often an important first step for solving a problem effectively. In sequential decision problems, Markov decision process (MDP) (Bellman 1957b; Puterman 1994) is a model formulation that has been commonly used, due to its generality, flexibility, and applicability to a wide range of problems. Despite these advantages, there are three necessary conditions that must be satisfied before the MDP model can be applied; that is,

1. The environment model is given in advance (a completely-known environment).
2. The environment states are completely observable (fully-observable states, implying a Markovian environment).
3. The environment parameters do not change over time (a stationary environment).

These prerequisites, however, limit the usefulness of MDPs. In the past, research efforts have been made towards relaxing the first two conditions, leading to different classes of problems as illustrated in Figure 1.

		Model of Environment: Known	Model of Environment: Unknown
States of Environment	Completely Observable	MDP	Traditional RL
	Partially Observable	Partially Observable MDP	Hidden-state RL

Fig. 1. Categorization into four related problems with different conditions. Note that the degree of difficulty increases from left to right and from upper to lower.

This paper mainly addresses the first and third conditions, whereas the second condition is only briefly discussed. In particular, we are interested in a special type of nonstationary environments that repeat their dynamics in a certain manner. We propose a formal model for such environments. We also develop algorithms for learning the model parameters and for computing optimal policies.

Before we proceed, let us briefly review the four categories of problems shown in Figure 1 and define the terminology that will be used in this paper.

1.1 Four Problem Types

Markov Decision Process

MDP is the central framework for all the problems we discuss in this section. An MDP formulates the interaction between an agent and its environment. The environment consists of a state space, an action space, a probabilistic state transition function, and a probabilistic reward function. The goal of the agent is to find, according to its optimality criterion, a mapping from states to actions (i.e. policy) that maximizes the long-term accumulated rewards. This policy is called an *optimal policy*. In the past, several methods for solving Markov decision problems have been developed, such as value iteration and policy iteration (Bellman 1957a).

Reinforcement Learning

Reinforcement learning (RL) (Kaelbling *et al.* 1996; Sutton and Barto 1998) is originally concerned with learning to perform a sequential decision task based only on scalar feedbacks, without any knowledge about what the correct actions should be. Around a decade ago researchers realized that RL problems could naturally be formulated into incompletely known MDPs. This realization is important because it enables one to apply existing MDP algorithms to RL problems. This has led to research on *model-based* RL. The model-based RL approach first reconstructs the environment model by collecting experience from its interaction with the world, and then applies conventional MDP methods to find a solution. On the contrary, *model-free* RL learns an optimal policy directly from the experience. It is this second approach that accounts for the major difference between RL and MDP algorithms. Since less information is available, RL problems are in general more difficult than the MDP ones.

Partially Observable Markov Decision Process

The assumption of having fully-observable states is sometimes impractical in the real world. Inaccurate sensory devices, for example, could make this condition difficult to hold true. This concern leads to studies on extending MDP to partially-observable MDP (POMDP) (Monahan 1982; Lovejoy 1991; White III 1991). A POMDP basically introduces two additional components to the original MDP, i.e. an observation space and an observation probability function.

Observations are generated based on the current state and the previous action, and are governed by the observation function. The agent is only able to perceive observations, but not states themselves. As a result, past observations become relevant to the agent's choice of actions. Hence, POMDPs are sometimes referred to as non-Markovian MDPs. Traditional approaches to POMDPs (Sondik 1971; Cheng 1998; Littman *et al.* 1995b; Cassandra *et al.* 1997; Zhang *et al.* 1997) maintain a probability distribution over the states, called *belief state*. It essentially transforms the problem into an MDP one with an augmented (and continuous) state space. Unfortunately, solving POMDP problems exactly is known to be intractable in general (Papadimitriou and Tsitsiklis 1987; Littman *et al.* 1995a).

Hidden-State Reinforcement Learning

Recently, research has been conducted on the case where the environment is both incompletely known and partially observable. This type of problems is sometimes referred to as hidden-state reinforcement learning, incomplete perception, perception aliasing, or non-Markovian reinforcement learning. Hidden-state RL algorithms can also be classified into model-based and model-free approaches. For the former, a variant of the Baum-Welch algorithm (Chrisman 1992) is typically used for model reconstruction, and hence turns the problem into a conventional POMDP. Optimal policies can then be computed by using existing POMDP algorithms. For the latter, research efforts are diverse, ranging from state-free stochastic policy (Jaakkola *et al.* 1995), to recurrent Q-learning (Schmidhuber 1991; Lin and Mitchell 1992), to finite-history-window approach (McCallum 1995; Lin and Mitchell 1992). Nevertheless, most of the model-free POMDP algorithms yield only sub-optimal solutions. Among the four classes of problems aforementioned, hidden-state RL problems are expected to be the most difficult.

1.2 Nonstationary Environments

Traditional MDP problems typically assume that environment dynamics (i.e., MDP parameters) are always fixed (i.e., *stationary*). This assumption, however, is not realistic in many real-world applications. In elevator control (Crites and Barto 1996), for example, the passenger arrival and departure rates can vary significantly over one day, and should not be modeled by a fixed MDP.

Previous studies on nonstationary MDPs (Puterman 1994) presume that changes of the MDP parameters are exactly known in every time step. Given this assumption, solving nonstationary MDP problems is trivial, as the problem can be recast into a stationary one (with a much larger state space) by performing state augmentation. Nevertheless, extending the idea to incompletely-known environmental changes (i.e., to the reinforcement learning framework) is far more difficult.

In fact, RL (Kaelbling *et al.* 1996; Sutton and Barto 1998) in nonstationary environments is an impossible task if there exists no regularity in the way environment dynamics change. Hence, some degree of regularity must be assumed.

Typically, nonstationary environments are presumed to change slowly enough such that on-line RL algorithms can be employed to keep track of the changes. The online approach is memoryless in the sense that even if the environment ever reverts to the previously learned dynamics, learning must still start all over again. There are a few heuristic approaches along this line (Littman and Ackley 1991; Sutton 1990; Sutton and Barto 1998).

1.3 The Properties of Our Proposed Model

Herein we propose a formal environment model (Choi *et al.* 1999) for the nonstationary environments that repeat their dynamics over time. Our model is inspired by observations from an interesting class of nonstationary RL tasks. Throughout this section we illustrate the properties of such nonstationary environments by using the elevator control problem as an example.

Property 1: A Finite Number of Environment Modes

The first property we observed is that environmental changes are confined to a finite number of *environment modes*. Modes are stationary environments that possess distinct environment dynamics and require different control policies. At any time instant, the environment is assumed to be in exactly one of these modes. This concept of modes seems to be applicable to many, though not all, real-world tasks. In the elevator control problem, a system might operate in a morning-rush-hour mode, an evening-rush-hour mode and a non-rush-hour mode. One can also imagine similar modes for other real-world control tasks, such as traffic control, dynamic channel allocation (Singh and Bertsekas 1996), and network routing (Boyan and Littman 1994).

Property 2: Partially Observable Modes

Unlike states, environment modes cannot be directly observed. Instead, the current mode can only be estimated according to the past state transitions. It is analogous to the elevator control example in that the passenger arrival rate and pattern can only be partially observed through the occurrence of pick-up and drop-off requests.

Property 3: Modes Evolving as a Markov Process

Normally, mode transitions are stochastic events and are independent of the control system's response. In the elevator control problem, the events that change the current mode of the environment could be an emergency meeting in the administrative office, or a tea break for the staff on the 10th floor. Obviously, the elevator's response has no control over the occurrence of these events.

Property 4: Infrequent Mode Transitions

Mode transitions are relatively infrequent. In other words, a mode is more likely to retain for some time before switching to another one. Take the emergency meeting as an example, employees on different floors take time to arrive at the administrative office, and thus would generate a similar traffic pattern (drop-off requests on the same floor) for some period of time.

Property 5: Small Number of Modes

It is common that, in many real-world applications, the number of modes is much fewer than the number of states. In the elevator control example, the state space comprises of all possible combinations of elevator positions, pick-up and drop-off requests, and certainly would be huge. On the other hand, the mode space could be small. For instance, an elevator control system can simply have the three modes as described above to approximate the reality.

Based on these properties, an environment model is now proposed. The whole idea is to introduce a mode variable to capture environmental changes. Each mode specifies an MDP and hence completely determines the current state transition function and reward function (property 1). A mode, however, is not directly observable (property 2), and evolves with time according to a Markov process (property 3). The model is therefore called *hidden-mode model*.

Note that the hidden-mode model does not impose any constraint to satisfy properties 4 and 5. In other words, the model is flexible enough to work for environments where these two properties do not hold. Nevertheless, as will be shown later, these properties can be utilized to help the learning in practice.

The hidden-mode model also has its limitations. For instance, one may argue that the mode of an environment should preferably be continuous. While this is true, for tractability, we assume the mode is discrete. This implies that our model, as for any other model, is only an abstraction of the real world. Moreover, we assume that the number of modes is known in advance. We will seek to relax these assumptions in future research.

1.4 Related Work

Our hidden-mode model is closely related to the nonstationary environment model proposed by Dayan and Sejnowski (1996). Although our model is more restrictive in terms of representational power, it involves much fewer parameters and is thus easier to learn. Besides, other than the number of possible modes that should be known in advance, we do not assume any other knowledge about the way environment dynamics change[1]. Dayan and Sejnowski, on the other hand, assume that one knows precisely how the environment dynamics change.

[1] That is, the transition probabilities of the Markov process governing mode changes, though fixed, are unknown in advance.

The hidden-mode model can also be viewed as a special case of the hidden-state model, or *partially observable Markov decision process* (POMDP). As will be shown later, a hidden-mode model can always be represented by a hidden-state model through state augmentation. Nevertheless, modeling a hidden-mode environment via a hidden-state model will unnecessarily increase the problem complexity. We discuss the conversion from the former to the latter in Section 2.2.

1.5 Our Focus

In order for RL to take place, one may choose between the model-based and model-free approaches. This paper is primarily concerned with the model-based approach, and concentrates on how a hidden-mode model can be learned based on the Baum-Welch algorithm. The issue of finding the optimal policy will only be addressed briefly.

1.6 Organization

The rest of this paper is organized as follows. In the next section, we describe the hidden-mode model by defining the hidden-mode Markov decision process (HM-MDP) and illustrate how it can be reformulated into a POMDP. Section 3 will subsequently discuss how a hidden-mode model can be learned in two different representations — a POMDP or an HM-MDP. A variant of the Baum-Welch algorithm for learning HM-MDP is proposed. These two approaches are then compared empirically in Section 4. In Section 5, we will briefly discuss how hidden-mode problems can be solved. Then we highlight the assumptions of our model and discuss its applicability in Section 6. Finally, Section 7 pinpoints some directions for future research and Section 8 summarizes our research work.

2 Hidden-Mode Markov Decision Processes

This section presents our hidden-mode model. Basically, a hidden-mode model is defined as a finite set of MDPs that share the same state space and action space, with possibly different transition functions and reward functions. The MDPs correspond to different modes in which a system operates. States are completely observable and their transitions are governed by an MDP. In contrast, modes are not directly observable and their transitions are controlled by a Markov chain. We refer to such a process as a *hidden-mode Markov decision process* (HM-MDP). Figure 2 gives an example of HM-MDP.

2.1 Formulation

Formally, an HM-MDP is defined as an 8-tuple $(Q, S, A, X, Y, R, \Pi, \Psi)$, where Q, S and A represent the sets of modes, states and actions respectively; the mode transition function X maps mode m to n with a fixed probability of x_{mn}; the

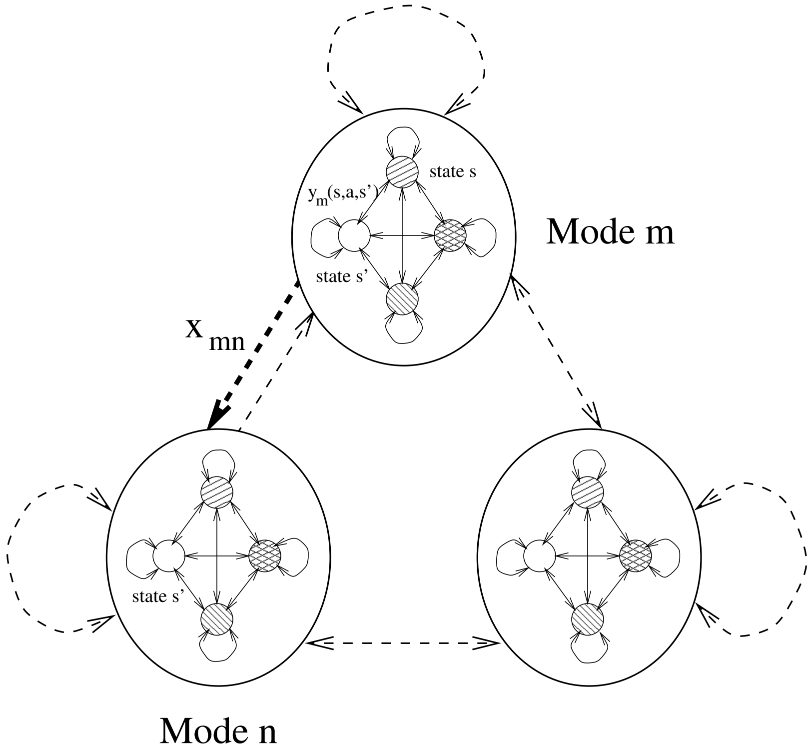


Fig. 2. A 3-mode, 4-state, 1-action HM-MDP. The values x_{mn} and $y_m(s, a, s')$ are the mode and state transition probabilities respectively.

state transition function Y defines transition probability, $y_m(s, a, s')$, from state s to s' given mode m and action a; the stochastic reward function R returns rewards with the mean value $r_m(s, a)$; Π and Ψ denote the prior probabilities of the modes and of the states respectively. The evolution of modes and states is depicted in Figure 3.

2.2 Reformulating HM-MDP as POMDP

HM-MDP is a subclass of POMDP. In other words, it is always possible to reformulate the former as a special case of the latter. In particular, one may take an ordered pair of any mode and observable state in the HM-MDP as a hidden state in a POMDP, and any observable state of the former as an observation of the latter. Suppose the observable states s and s' are in modes m and n respectively. These two HM-MDP states together with their corresponding modes form two hidden states $\langle m, s\rangle$ and $\langle n, s'\rangle$ for its POMDP counterpart. The transition probability from $\langle m, s\rangle$ to $\langle n, s'\rangle$ is then simply the mode transition probability x_{mn} multiplied by the state transition probability $y_m(s, a, s')$. For an M-mode,

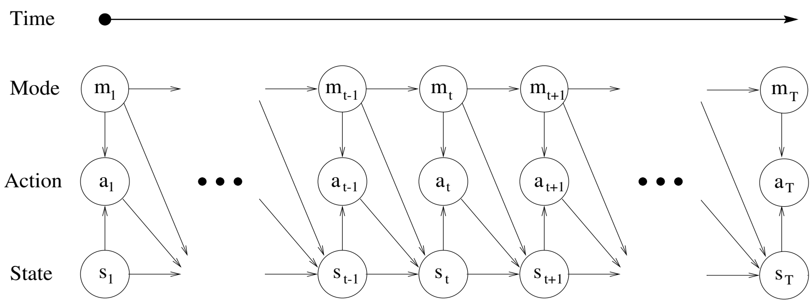


Fig. 3. The evolution of an HM-MDP. Each node represents a mode, action or state variable. The arcs indicate dependencies between the variables.

N-state, K-action HM-MDP, the equivalent POMDP thus has N observations and MN hidden states. A formal reformulation is detailed in Figure 4.

$$POMDP = (S', A', T', R', Z', Q', \Pi')$$

$$S' = Q \times S, \quad Z' = S, \quad A' = A$$

$$T' : \{p_{ij}(a) = x_{mn} \cdot y_m(s, a, s') \mid i = \langle m, s \rangle \in S', \; j = \langle n, s' \rangle \in S'\}$$

$$Q' : \{q_i(a, s') = \begin{cases} 1 & : \text{ if } s = s' \\ 0 & : \text{ if } s \neq s' \end{cases} \mid i = \langle m, s \rangle \in S', \quad s' \in Z'\}$$

$$R' : \{r_i(a) = r_m(s, a) \mid i = \langle m, s \rangle \in S'\}$$

$$\Pi' = \{\pi'_i = \pi_m \cdot \psi_s \mid i = \langle m, s \rangle \in S'\}$$

Fig. 4. Reformulating HM-MDP into POMDP

Note that $S', A', T', R', O', Q', \Pi'$ are the state space, action space, state transition function, reward function, observation space, observation function, and prior state probabilities of the resulting POMDP respectively. The following lemmas prove that HM-MDPs are indeed a subclass of POMDPs.

Lemma 1. *The HM-MDP to POMDP transformation satisfies the state transition function requirement, namely* $\sum_j p_{ij}(a) = 1$.

Proof: From the transformation, we know that

$$\begin{aligned}
\sum_j p_{ij}(a) &= \sum_{\langle n,s'\rangle \in Q' \times S'} p_{\langle m,s\rangle,\langle n,s'\rangle}(a) \\
&= \sum_{n \in Q'} \sum_{s' \in S'} x_{mn} \cdot y_m(s,a,s') \\
&= \sum_{n \in Q'} x_{mn} \cdot \sum_{s' \in S'} y_m(s,a,s') \\
&= \sum_{n \in Q'} x_{mn} \cdot 1 \\
&= 1
\end{aligned}$$

□

Lemma 2. *HM-MDPs form a subclass of POMDPs.*

Proof: Given an HM-MDP and its corresponding POMDP, there is a 1-1 mapping from the mode-state pairs of the HM-MDP into the states of the POMDP. This implies that for every two different pairs of mode and state $\langle m, s\rangle$ and $\langle m', s'\rangle$, there are corresponding distinct states in the POMDP. The transformation ensures that the two states are the same if and only if $m = m'$ and $s = s'$. It also imposes the same transition probability and average reward between two mode-state pairs and for the corresponding states in the POMDP. It implies that the transformed model is equivalent to the original one. Therefore HM-MDPs are a subclass of POMDPs. □

Figure 5 demonstrates the reformulation for a 3-mode, 4-state, 1-action HM-MDP and compares the model complexity with its equivalent POMDP. Note that most state transition probabilities (dashed lines) in the POMDP are collapsed into mode transition probabilities in the HM-MDP through parameter sharing. This saving is significant. In the example, the HM-MDP model has a total of 57 transition probabilities, while its POMDP counterpart has 144. In general, an HM-MDP contains much fewer parameters ($N^2MK + M^2$) than its corresponding POMDP (M^2N^2K).

3 Learning a Hidden-Mode Model

Now it becomes clear that there are two ways to learn a hidden-mode model: either learning an HM-MDP directly, or learning an equivalent POMDP. In this section, we first briefly discuss how the latter can be achieved by a variant of the Baum-Welch algorithm, and then develop a similar algorithm for HM-MDP.

3.1 Learning a POMDP Model

Traditional research in POMDP (Monahan 1982; Lovejoy 1991) assumes a known environment model and is concerned with finding an optimal policy. Chrisman (1992) was the first to study the learning of POMDP models from experience.

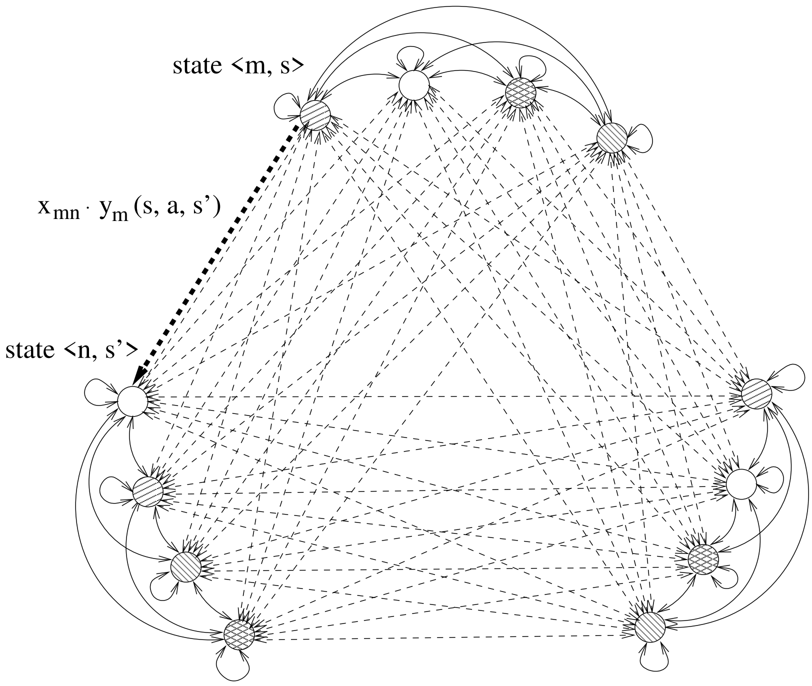


Fig. 5. Reformulating a 3-mode, 4-state, 1-action HM-MDP described in Figure 2 as an equivalent POMDP. For brevity, exact transition probability values are not shown. State s of mode m in Figure 2 is relabeled as $<m, s>$. Note that HM-MDP has much fewer model parameters than its POMDP counterpart.

Chrisman's work is based on the Baum-Welch algorithm, which was originally proposed for learning hidden Markov models (HMMs) (Rabiner 1989). Based on the fact that a POMDP can be viewed as a collection of HMMs, Chrisman proposed a variant of the Baum-Welch algorithm for POMDP. This POMDP Baum-Welch algorithm requires $\Theta(M^2N^2T)$ time and $\Theta(M^2N^2K)$ storage for learning an M-mode, N-state, K-action HM-MDP, given T data items. However, Chrisman's algorithm does not learn the reward function. One possible extension is to estimate the reward function by averaging the obtained rewards, weighted by the estimated state certainty. The effectiveness of the algorithm will be examined in the next section.

3.2 HM-MDP Baum-Welch Algorithm

We now extend the Baum-Welch algorithm to the learning of an HM-MDP model. The outline of the algorithm, called HM-MDP Baum-Welch, is shown in Figure 6. This new algorithm is similar to the POMDP version, except that the auxiliary variables are redefined.

Given a collection of data and an initial model parameter vector $\bar{\theta}$.
repeat
  $\theta = \bar{\theta}$
  Compute forward variables α_t. (Figure 7)
  Compute backward variables β_t. (Figure 8)
  Compute auxiliary variables ξ_t and γ_t. (Figure 9)
  Compute the new model parameter $\bar{\theta}$. (Figure 10)
until $\max_i |\bar{\theta}_i - \theta_i| < \epsilon$

Fig. 6. The skeleton of HM-MDP Baum-Welch algorithm

The intuition behind both algorithms remains the same; i.e., estimating the parameters of the hidden variables by counting their expected number of transitions. This counting, however, can only be inferred from the observations by maintaining a set of auxiliary variables. Suppose that the number of transitions from the hidden variable i to j is known, the transition probability from i to j can then be computed by the following equation:

$$\Pr(j|i) = \frac{\text{the number of transitions from } i \text{ to } j}{\text{the number of visits to } i}$$

where the denominator is simply the numerator summing over all j.

The central problem now becomes how to count the transitions of the hidden variables. While its exact value is unknown, it is possible to estimate the expected value through inferences on the observation sequence. We define $\xi_t(i, j)$ as the expected number of state transitions from i to j at time step t. Given a sequence of T observations, i.e., $T - 1$ transitions, the total number of transitions from state i to j becomes $\sum_{t=1}^{T} \xi_t(i, j)$, and the total number of visits to i is $\sum_j \sum_{t=1}^{T} \xi_t(i, j)$.

Thus far, our algorithm is not different from the standard Baum-Welch algorithm. The key difference between the two algorithms comes from the inference part (i.e., maintaining the auxiliary variables). As the observed state sequence is shifted one step forward in our model, additional attention is needed for handling the boundary cases. In the subsequent sections, the intuitive meanings and definitions of the auxiliary variables will be given.

Computing Forward Variables

Let the collection of data D be denoted as $(\mathsf{S}, \mathsf{A}, \mathsf{R})$, where S is the observation sequence $s_1 \, s_2 \cdots s_T$, A the random[2] action sequence $a_1 \, a_2 \cdots a_T$, and R the reward

[2] It is worth mentioning that although the optimal action is a function of modes and states, the choice of action should be at random in the model-learning phase, or called exploration phase.

sequence $r_1 r_2 \cdots r_T$. We define forward variables $\alpha_t(i)$ as $\Pr(s_1, s_2, ..., s_t, m_{t-1} = i|\theta, \mathsf{A})$, i.e. the joint probability of observing the partial state sequence from s_1 up to s_t and the mode at time $t-1$ being in i, given the model θ and the random action sequence A. To compute the forward variables efficiently, a dynamic programming approach is depicted in Figure 7.

Computing Forward Variables:

for all $i \in Q$,
$\alpha_1(i) = \psi_{s_1}$

for all $i \in Q$,
$\alpha_2(i) = \pi_i \; \psi_{s_1} \; y_i(s_1, a_1, s_2)$

for all $j \in Q$,
$\alpha_{t+1}(j) = \sum_{i \in Q} \alpha_t(i) \, x_{ij} \, y_j(s_t, a_t, s_{t+1})$

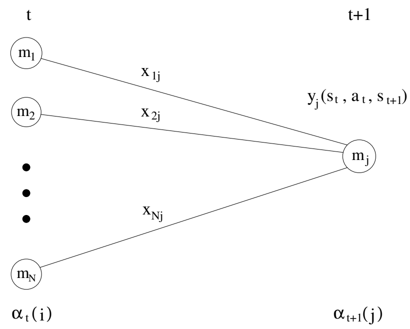


Fig. 7. Computing forward variables

Computing Backward Variables

Backward variable $\beta_t(i)$ denotes the probability $\Pr(s_{t+1}, s_{t+2}, \cdots, s_T|s_t, m_{t-1} = i, \theta, \mathsf{A})$, which represents the probability of obtaining the partial state sequence s_{t+1}, s_{t+2} up to s_T, given the state s_t, the mode being i at time $t-1$, the model θ, and the random action sequence A. Unlike forward variables, backward variables are computed backward from β_T. The computational steps required for the backward variables are illustrated in Figure 8.

Computing Auxiliary Variables

With the forward and backward variables, two additional auxiliary variables $\xi_t(i,j)$ and $\gamma_t(i)$ can now be defined. Formally, $\xi_t(i,j)$ is defined as $\Pr(m_{t-1} = i, m_t = j|\mathsf{S}, \mathsf{A}, \theta)$, i.e. the probability of being in mode i at time $t-1$ and in mode j at time t, given the state and action sequences, and the model θ. Figure 9 depicts how the forward and backward variables can be combined to compute $\xi_t(i,j)$.

Auxiliary variable $\gamma_t(i)$ is defined as $\Pr(m_t = i|\mathsf{S}, \mathsf{A}, \theta)$, i.e. the probability of being in mode i at time t, given the state and action sequences, and the model θ. Then $\gamma_t(i)$ is simply the sum of $\xi_{t+1}(i,j)$ over all possible resultant modes j; that is, Computing Backward Variables:

for all $i \in Q$,
$\beta_T(i) = 1$

for all $i \in Q$,
$\beta_t(i) = \sum_{j \in Q} x_{ij}\ y_j(s_t, a_t, s_{t+1})\ \beta_{t+1}(j)$

for all $i \in Q$,
$\beta_1(i) = \sum_{j \in Q} \pi_j\ y_j(s_1, a_1, s_2)\ \beta_2(j)$

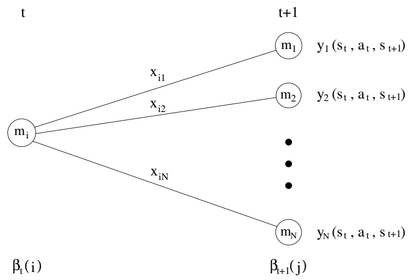
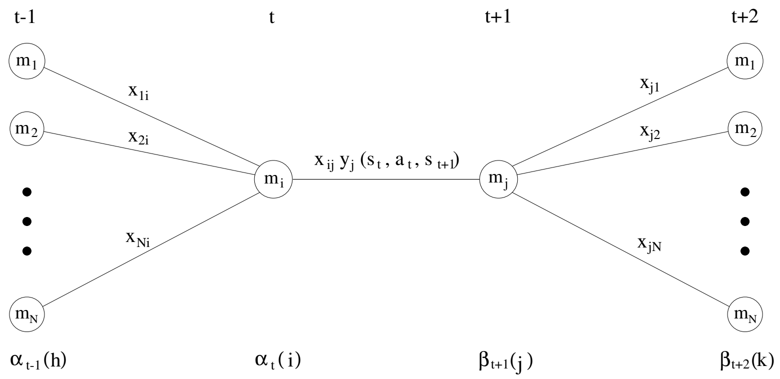


Fig. 8. Computing backward variables



$$\gamma_t(i) = \sum_{j=1}^{N} \xi_{t+1}(i, j)$$

Parameter Reestimation

With these auxiliary variables, the model parameters can be updated. In particular, the prior mode distribution π_i, i.e. the probability of being in mode i at time $t = 1$, is simply the auxiliary variable $\gamma_1(i)$.

The mode transition probability x_{ij} is the number of mode transitions from i to j divided by the number of visits to mode i. Although these two values cannot be known exactly, they can be estimated by summing up the variables ξ and γ respectively:

$$\bar{x}_{ij} = \frac{\text{expected number of mode transitions from } i \text{ to } j}{\text{expected number of visits to mode } i}$$

Computing Auxiliary Variables:
For all $i, j \in Q$,

$$\xi_t(i,j) = \frac{\alpha_t(i)\, x_{ij}\, y_j(s_t, a_t, s_{t+1})\, \beta_{t+1}(j)}{\sum_{k \in Q} \alpha_T(k)}$$

For all $i \in Q$,

$$\gamma_t(i) = \sum_{j \in Q} \xi_{t+1}(i,j)$$

Fig. 9. Computing auxiliary variables

$$= \frac{\sum_{t=2}^{T} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

Similarly, the state transition probabilities and reward function can be reestimated by the following:

$$\begin{aligned} \bar{y}_i(j,k,l) &= \frac{\text{expected number of visits to mode } i \text{ state } j \text{, with action } k \text{ and resulting in } l}{\text{expected number of visits to mode } i \text{ state } j \text{, with action } k} \\ &= \frac{\sum_{t=1}^{T-1} \gamma_t(i)\, \delta(s_t, j)\, \delta(a_t, k)\, \delta(s_{t+1}, l)}{\sum_{h \in S} \sum_{t=1}^{T-1} \gamma_t(i)\, \delta(s_t, j)\, \delta(a_t, k)\, \delta(s_{t+1}, h)} \end{aligned}$$

$$\begin{aligned} \bar{r}_i(j,k) &= \frac{\text{expected reward received by taking action } k \text{ at mode } i \text{ state } j}{\text{expected number of visits to mode } i \text{ state } j \text{ with action } a} \\ &= \frac{\sum_{t=1}^{T-1} \gamma_t(i)\, \delta(a_t, k)\, \delta(s_t, j)\, r_t}{\sum_{t=1}^{T-1} \gamma_t(i)\, \delta(a_t, k)\, \delta(s_t, j)} \end{aligned}$$

The function $\delta(a, b)$ is defined as 1 if $a = b$, or 0 otherwise. It is used for selecting the particular state and action. For a small data set, it is possible that some states or actions do not occur at all. In that case, the denominators of the formulae are equal to zero, and the parameters should remain unchanged. Figure 10 summarizes the parameter reestimation procedure.

$$\bar{x}_{ij} = \begin{cases} \dfrac{\sum_{t=2}^{T} \xi_t(i,j)}{\sum_{t=1}^{T} \gamma_t(i)} & : \text{ if denominator} \neq 0 \\ x_{ij} & : \text{ otherwise} \end{cases}$$

$$\bar{y}_i(j,k,l) = \begin{cases} \dfrac{\sum_{t=1}^{T-1} \gamma_t(i)\, \delta(s_t, j)\, \delta(s_{t+1}, l)\, \delta(a_t, k)}{\sum_{l \in S} \sum_{t=1}^{T-1} \gamma_t(i)\, \delta(s_t, j)\, \delta(s_{t+1}, l)\, \delta(a_t, k)} & : \text{ if denominator} \neq 0 \\ y_i(j,k,l) & : \text{ otherwise} \end{cases}$$

$$\bar{r}_i(j,k) = \begin{cases} \dfrac{\sum_{t=1}^{T-1} \gamma_t(i)\, \delta(a_t, k)\, \delta(s_t, j)\, r_t}{\sum_{t=1}^{T-1} \gamma_t(i)\, \delta(a_t, k)\, \delta(s_t, j)} & : \text{ if denominator} \neq 0 \\ r_i(j,k) & : \text{ otherwise} \end{cases}$$

$$\bar{\pi}_i = \gamma_1(i)$$

Fig. 10. Parameter reestimation for HM-MDP Baum-Welch algorithm

It is not difficult to see that the HM-MDP Baum-Welch algorithm requires only $\Theta(M^2T)$ time and $\Theta(MN^2K + M^2)$ storage, which gives a significant reduction when compared with $\Theta(M^2N^2T)$ time and $\Theta(M^2N^2K)$ storage in the POMDP approach.

4 Empirical Studies

This section empirically examines the POMDP Baum-Welch[3] and HM-MDP Baum-Welch algorithms in terms of the required data size and time. Experiments on various model sizes and settings were conducted. The results are quite consistent. In the following, some details of a typical run are presented for illustration.

4.1 Experimental Setting

The experimental model is a randomly generated HM-MDP with 3 modes, 10 states and 5 actions. Note that this HM-MDP is equivalent to a fairly large

[3] Chrisman's algorithm also attempts to learn a minimal possible number of states. Here we are concerned only with learning of the model parameters.

POMDP, with 30 hidden states, 10 observations and 5 actions. In order to simulate the infrequent mode changes, each mode is set to have a minimum probability of 0.9 in looping back to itself. In addition, each state of the HM-MDP has 3 to 8 non-zero transition probabilities, and rewards are uniformly distributed between $r_m(s, a) \pm 0.1$. This reward distribution, however, is not disclosed to the learning agents.

The experiments were run with data of various sizes, using the same initial model. The model was also randomly generated in the form of HM-MDP. To ensure fairness, the equivalent POMDP model was used for POMDP Baum-Welch learning. For each data set, the initial model was first loaded, and the selected algorithm iterated until the maximum change of the model parameters was less than a threshold of 0.0001. After the algorithm terminated, the model learning time was recorded, and the model estimation errors were computed. The experiment was then repeated for 11 times with different random seeds in order to compute the median.

4.2 Performance Measure

The HM-MDP and POMDP Baum-Welch algorithms learn a hidden-mode model in different representations. To facilitate comparison, all models were first converted into POMDP form. Model estimation errors can then be measured in terms of the minimum difference between the learned model and the actual model. As the state indices for the learned model might be different from the actual one, a renumbering of the state indices is needed. In our experiment, an indexing scheme that minimizes the sum of the squares of differences on the state transition probabilities between the learned and the actual models was used (provided the constraints on the observation probabilities are preserved). Figure 11 (a) and (b) report respectively the sum of the squares of differences on the transition function and on the reward function using this indexing scheme.

Regarding the computational requirement of the algorithms, the total CPU running time was measured on a SUN Ultra I workstation. Table 1 reports the model learning time in seconds.

Approach	Data Set Size				
	1000	2000	3000	4000	5000
HM-MDP	4.60	18.72	15.14	9.48	10.07
POMDP	189.40	946.78	2164.20	3233.56	4317.19

Table 1. CPU time in seconds

4.3 Empirical Results

Conclusion can now be drawn. Generally speaking, both algorithms can learn a more accurate environment model as the data size increases (Figure 11). This result is not surprising since both algorithms are statistically based, and hence their performances rely largely on the amount of data provided. When the training data size is too small (less than 1000 in this case), both algorithms perform about equally poorly. However, as the data size increases, HM-MDP Baum-Welch improves substantially faster than POMDP Baum-Welch.

Our experiment reveals that HM-MDP Baum-Welch was able to learn a fairly accurate environment model with a data size of 2500. POMDP Baum-Welch, on the contrary, needs a data size of 20000 (not shown) in order to achieve a comparable accuracy. In fact, in all the experiments we conducted, HM-MDP Baum-Welch always required a much smaller data set than the POMDP Baum-Welch. We believe that this result holds in general because in most cases, an HM-MDP consists of fewer free parameters than its POMDP counterpart.

In terms of computational requirement, HM-MDP Baum-Welch is much faster than POMDP Baum-Welch (Table 1). We believe this is also true in general for the same reason described above. In addition, computational time is not necessarily monotonically increasing with the data size. It is because the total computational requirements depend not only on the data size, but also on the number of iterations being executed. From our experiments, we notice that the number of iterations tends to decrease as the data size increases.

5 Solving Hidden-Mode Problems

In this section we describe briefly how hidden-mode problems can be solved. Since HM-MDPs are a special class of POMDPs, a straightforward approach is first to convert a hidden-mode problem into a POMDP one and subsequently to apply POMDP algorithms. Nevertheless, POMDP algorithms do not exploit the special structures of HM-MDPs. Herein, we develop a value iteration approach for HM-MDPs[4]. The main idea of the algorithm is to exploit the HM-MDP structure by decomposing the value function based on the state space. This algorithm is akin to the POMDP value iteration conceptually but is much more efficient.

5.1 Value Iteration for HM-MDPs

Many POMDP algorithms maintain a probability distribution over the state space known as *belief state*. As a result, a policy is a mapping from belief states to actions. In HM-MDPs, a probability distribution over the mode space is maintained. We name it *belief mode*. Unlike the POMDP approach, the belief mode divides the belief states into an observable part (i.e. the observable states) and an unobservable part (i.e. the hidden modes). This representation minimizes the number of hidden variables. For every observed state and action, the belief mode b is updated as follows.

[4] A more detailed description of the algorithm can be found in (Choi 2000).

$$b^a_{s'}(m') = \frac{\sum_{m\in Q}\Pr(m'|m)\cdot\Pr(s'|m,s,a)\cdot b(m)}{\sum_{m'\in Q}\sum_{m\in Q}\Pr(m'|m)\cdot\Pr(s'|m,s,a)\cdot b(m)}$$
$$= \frac{\sum_{m\in Q} x_{m'm}\cdot y_m(s,a,s')\cdot b(m)}{\sum_{m'\in Q}\sum_{m\in Q} x_{m'm}\cdot y_m(s,a,s')\cdot b(m)}$$

where $b^a_{s'}(m')$ is the probability of being in mode m', given the action a and the state s'. The numerator computes the likelihood of the next mode and the denominator is the normalization factor.

An HM-MDP policy now becomes a mapping from belief modes and states to actions. Specifically, an optimal action for a belief mode b and the current state s maximizes the following value function:

$$V(b,s) = \max_{a\in A}\Big(\sum_{m\in Q} r_m(s,a)\cdot b(m) + \gamma\sum_{s'\in S}\Pr(s'|b,s,a)V(b^a_{s'},s')\Big) \qquad (1)$$

where $r_m(s,a)$ is the reward function and γ is the discount factor. Since s is a discrete variable, one can view the value function V as $|S|$ value functions. Note that these decomposed value functions are still piecewise linear and convex. One can therefore use Equation (1) for the value iteration to compute the optimal value function for each state in S.

5.2 Empirical Results

We implement Equation (1) by using a variant of incremental pruning (Zhang and Liu 1997; Cassandra *et al.* 1997) and run the program on a SPARC Ultra 2 machine. A number of experiments were conducted based on some simplified real-world problems. The descriptions of the problems can be found in (Choi 2000). Table 2 gives a summary of the tasks. In these experiments, a discount factor of 0.95 and a solution tolerance of 0.1 were used. Table 3 reports the CPU time in seconds, the number of vectors in the resulting value functions, and the number of required epochs. While incremental pruning is considered as one of the most efficient POMDP algorithms, our experiment showed that it is unable to solve any of these problems within the specified time limit.

There are two main reasons why the HM-MDP approach is more efficient than the POMDP one. First, the dimension of the vector is significantly reduced from $|Q|\cdot|S|$ to $|Q|$. Second, the most time-consuming part of the algorithm, namely the cross sum operation, no longer needs to be performed on the whole value function due to decomposition of the value function.

6 Discussions

The usefulness of a model depends on the validity of the assumptions made. In this section, we revisit the assumptions of HM-MDP, discuss the issues involved,

Table 2. HM-MDP problems

Problem	Modes	States	Actions
Traffic Light	2	8	2
Sailboat	4	16	2
Elevator	3	32	3

Table 3. Solving HM-MDP problems by using incremental pruning

Problem	POMDP Approach			HM-MDP Approach		
	Time	Vectors	Epochs	Time	Vectors	Epochs
Traffic Light	>259200	-	-	4380	404	114
Sailboat	>259200	-	-	170637	1371	112
Elevator	>259200	-	-	186905	3979	161

and shed some light on its applicability to real-world nonstationary tasks. Some possible extensions are also discussed.

A Finite Number of Environment Modes

MDP is a flexible framework that has been widely adopted in various applications. Among these there exist many tasks that are nonstationary in nature and are more suitable to be characterized by several, rather than a single, MDPs. The introduction of distinct MDPs for modeling different modes of the environment is a natural extension to those tasks.

One advantage of having distinct MDPs is that the learned model is more comprehensible: each MDP naturally describes a mode of the environment. In addition, this formulation facilitates the incorporation of prior knowledge into the model initialization step.

Partially Observable Modes

While modes are not directly observable, they may be estimated by observing the past state transitions. It is a crucial, and fortunately still reasonable, assumption that one needs to make.

Although states are assumed to be observable, it is possible to extend the model to allow partially observable states, i.e., to relax the second condition mentioned in Section 1. In this case, the extended model would be equivalent in representational power to a POMDP. This could be proved by showing the reformulation of the two models in both directions.

Modes Evolving as a Markov Process

This property may not always hold for all real-world tasks. In some applications, such as learning in a multi-agent environment or performing tasks in an adversary environment, the agent's actions might affect the state as well as the environment mode. In that case, an MDP instead of a Markov chain should be used to govern the mode transition process. Obviously, the use of a Markov chain involves fewer parameters and is thus preferable whenever possible.

Infrequent Mode Transitions

This is a property that generally holds in many applications. In order to characterize this property, a large transition probability for a mode looping back to itself can be used. Note that this is introduced primarily from a practical point of view, but is not a necessary condition for our model. In fact, we have tried to apply our model-learning algorithms to problems in which this property does not hold. We find that our model still outperforms POMDP, although the required data size is typically increased for both cases.

Using high self-transition probabilities to model rare mode changes may not always be the best option. In some cases mode transitions are also correlated with the time of a day (e.g. busy traffic in lunch hours). In this case, time (or the mode sequence) should be taken into account for identifying the current mode. One simple way to model this property is to strengthen left-to-right transitions between modes, as in the left-to-right HMMs.

Small Number of Modes

This nice property significantly reduces the number of parameters in HM-MDP compared to that in POMDP, and makes the former more applicable to real-world nonstationary tasks.

The number of states can be determined by the learning agent. States can be distinguished by, for instance, transition probabilities, mean rewards, or utilities. McCallum (1995) has detailed discussions on this issue.

Likewise, the number of modes can be defined in various ways. After all, modes are used to discern changes of environment dynamics from noise. In practice, introducing a few modes is sufficient for boosting the system performance. More modes might help further, but not necessarily significantly. A trade-off between performance and response time must thus be decided. In fact, determining the optimal number of modes is an important topic that deserves further studies.

7 Future Work

There are a number of issues that need to be addressed in order to broaden the applicability of HM-MDPs. First, the number of modes is currently assumed to be known. In some situations, choosing the right number of modes can be difficult. Hence, we are now investigating the possibility of using Chrisman's or McCallum's hidden-state-splitting techniques (Chrisman 1992; McCallum 1993) to remove this limitation. Next, the problem-solving algorithm we presented here is preliminary. Further improvement, such as incorporating the point-based improvement technique (Zhang *et al.* 1999), can be achieved. We are also investigating an algorithm that further exploits the characteristics of the HM-MDP. We will present this algorithm in a separate paper. Finally, the exploration-exploitation issue is currently ignored. In our future work, we will address this important issue and apply our model to real-world nonstationary tasks.

8 Summary

Making sequential decisions in nonstationary environments is common in real-world problems. In this paper we presented a formal model, called hidden-mode Markov decision process, for a broad class of nonstationary sequential decision tasks. The proposed model is based on five properties observed in a special type of nonstationary environments, and is applicable to many traffic control type problems. Basically, the hidden-mode model is defined as a fixed number of partially observable modes, each of which specifies an MDP. While state and action spaces are fixed across modes, the transition and reward functions may differ according to the mode. In addition, the mode evolves according to a Markov chain.

HM-MDP is a generalization of MDP. In addition to the basic MDP characteristics, HM-MDP also allows the model parameters to change probabilistically. This feature is important because many real-world tasks are nonstationary in nature and cannot be represented accurately by a fixed model. Nevertheless, the hidden-mode model also adds uncertainty to the model parameters and makes the problem, in general, more difficult than the MDPs.

HM-MDP is a specialization of POMDP; it can always be transformed into a POMDP with an augmented state space. While POMDPs are superior in terms of representational power, HM-MDPs require fewer parameters, and therefore can provide a more natural formulation for certain type of nonstationary problems. Our experiments also show that this simplification significantly speeds up both the model-learning and the problem-solving procedures of HM-MDPs.

9 Acknowledgment

This research work is supported by Hong Kong Research Grants Council Grant: HKUST6152/98E.

References

R. E. Bellman, (1957). *Dynamic Programming.* Princeton University Press, Princeton, NJ.

R. E. Bellman, (1957). A Markovian decision process. *Journal of Mathematics and Mechanics*, 6:679–684.

J. A. Boyan and M. L. Littman, (1994). Packet routing in dynamically changing networks: a reinforcement learning approach. In *Advances in Neural Information Processing Systems 6*, pages 671–678, San Mateo, California. Morgan Kaugmann.

A. R. Cassandra, M. L. Littman, and N. Zhang, (1997) Incremental pruning: A simple, fast, exact algorithm for partially observable Markov decision processes. In *Uncertainty in Artificial Intelligence*, Providence, RI.

H.-T. Cheng, (1988). *Algorithms for Partially Observable Markov Decision Processes.* PhD thesis, University of British Columbia, British Columbia, Canada.

S. P. M. Choi, (2000). *Reinforcement Learning in Nonstationary Environments.* PhD thesis, Hong Kong University of Science and Technology, Department of Computer Science, HKUST, Clear Water Bay, Hong Kong, China, Jan.

S. P. M. Choi, D. Y. Yeung, and N. L. Zhang, (1999). An environment model for nonstationary reinforcement learning. In *Advances in Neural Information Processing Systems 12.* To appear.

L. Chrisman, (1992). Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *AAAI-92.*

R. H. Crites and A. G. Barto, (1996). Improving elevator performance using reinforcement learning. In D. Touretzky, M. Mozer, and M. Hasselmo, editors, *Advances in Neural Information Processing Systems 8.*

P. Dayan and T. J. Sejnowski, (1996). Exploration bonuses and dual control. *Machine Learning*, 25(1):5–22, Oct.

T. Jaakkola, S. P. Singh, and M. I. Jordan, (1995). Monte-Carlo reinforcement learning in non-Markovian decision problems. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, MA. The MIT Press.

L. P. Kaelbling, M. L. Littman, and A. W. Moore, (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, May.

L. J. Lin and T. M. Mitchell, (1992). Memory approaches to reinforcement learning in non-Markovian domains. Technical Report CMU-CS-92-138, Carnegie Mellon University, School of Computer Science.

M. L. Littman, A. R. Cassandra, and L. P. Kaelbling, (1995a). Learning policies for partially observable environments: Scaling up. In A. Prieditis and S. Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning*, pages 362–370, San Francisco, CA. Morgan Kaufmann.

M. L. Littman, A. R. Cassandra, and L. P. Kaelbling, (1995b). Efficient dynamic-programming updates in partially observable Markov decision processes. Technical Report TR CS-95-19, Department of Computer Science, Brown University, Providence, Rhode Island 02912, USA.

M. L. Littman and D. H. Ackley, (1991). Adaptation in constant utility non-stationary environments. In R. K. Belew and L. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 136–142, San Mateo, CA, Dec. Morgan Kaufmann.

W. S. Lovejoy, (1991). A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research*, 28:47–66.

A. McCallum, (1993). Overcoming incomplete perception with utile distinction memory. In *Tenth International Machine Learning Conference*, Amherst, MA.

A. McCallum, (1995). *Reinforcement Learning with Selective Perception and Hidden State.* PhD thesis, University of Rochester, Dec.

G. E. Monahan, (1982). A survey of partially observable Markov decision processes: Theory, models and algorithms. *Management Science*, 28:1–16.

C. H. Papadimitriou and J. N. Tsitsiklis (1987). The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450.

M. L. Puterman (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons.

L. R. Rabiner, (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2), Feb.

J. H. Schmidhuber (1990). Reinforcement learning in Markovian and non-Markovian environments. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems*, volume 3, pages 500–506, San Mateo, CA. Morgan Kaufmann.

S. Singh and D. P. Bertsekas, (1997). Reinforcement learning for dynamic channel allocation in cellular telephone systems. In *Advances in Neural Information Processing Systems 9*, 1997.

E. J. Sondik, (1971). *The Optimal Control of Partially Observable Markov Processes*. PhD thesis, Stanford University, Stanford, California, USA.

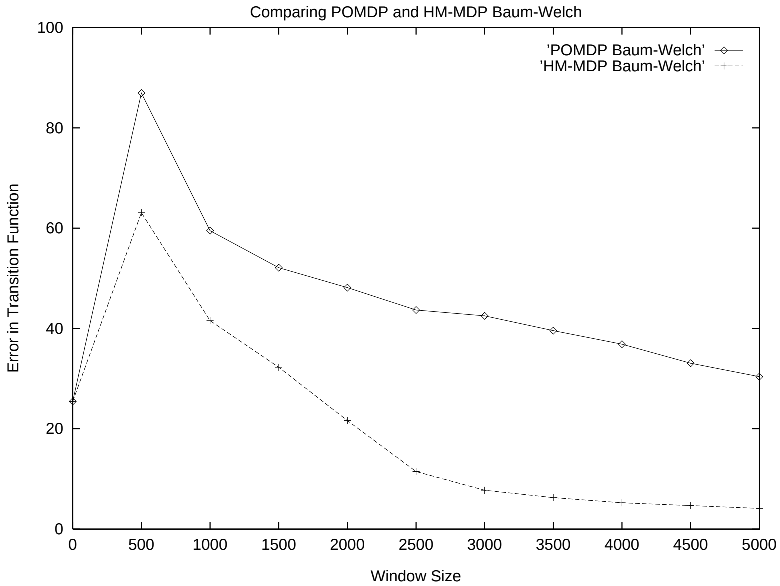
R. S. Sutton, (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224. Morgan Kaufmann.

R. S. Sutton and A. G. Barto, (1998). *Reinforcement Learning: An Introduction*. The MIT Press.

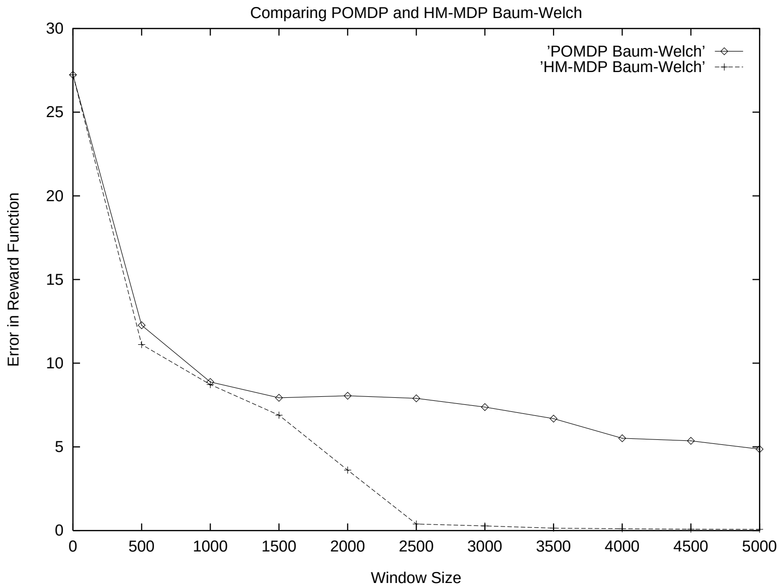
C. C. White III, (1991). Partially observed markov decision processes: A survey. *Annals of Operations Research*, 32.

N. L. Zhang, S. S. Lee, and W. Zhang, (1999). A method for speeding up value iteration in partially observable markov decision processes. In *Proceeding of 15th Conference on Uncertainties in Artificial Intelligence*.

N. L. Zhang and W. Liu, (1997). A model approximation scheme for planning in partially observable stochastic domains. *Journal of Artificial Intelligence Research*, 7:199 – 230.



(a) Error in Transition Function



(b) Error in Reward Function

Fig. 11. Model learning errors in terms of the transition probabilities and rewards. All environment models are in their POMDP form for comparison. The errors are measured by summing the squares of differences on the state transition probabilities and on the reward function respectively.