A (Revised) Survey of Approximate Methods for Solving Partially Observable Markov Decision Processes

Douglas Aberdeen
National ICT Australia,
Canberra, Australia.

December 8, 2003

Abstract

Partially observable Markov decision processes (POMDPs) are interesting because they provide a general framework for learning in the presence of multiple forms of uncertainty. We survey methods for learning within the POMDP framework. Because exact methods are intractable we concentrate on approximate methods. We explore two versions of the POMDP training problem: learning when a model of the POMDP is known, and the much harder problem of learning when a model is not available. The methods used to solve POMDPs are sometimes referred to as reinforcement learning algorithms because the only feedback provided to the agent is a scalar reward signal at each time step.

1 Introduction

Partially observable Markov decision processes (POMDPs) provide a framework for agents that learn how to act in their environment, or *world*. The POMDP model seamlessly incorporates uncertainty about an agent's perceptions, actions, and feedback. The only performance feedback given to agents is a scalar reward generated by the world at each time step. The learning problem is complicated by rewards which may be delayed from the actions that caused them, creating a *temporal credit assignment* problem.

In the unrestricted POMDP setting a number of complexity results have shown that computing optimal policies is intractable, thus most of the literature is devoted to computing approximate policies. We discuss the complexity of exact POMDP methods and present the main approaches to approximating POMDPs solutions using only a small amount of mathematics.

The following sections introduce the concepts of POMDPs, memory states, long-term values, and methods for solving fully observable MDPs. Section 6 discusses exact and approximate methods for solving POMDPs when the underlying POMDP parameters are known. Section 7 assumes that these parameters are *not* known, making the task extremely difficult. Section 8 describes miscellaneous methods useful in the modelled and model free domains. They include variance reduction methods, multi-agent settings and hierarchical POMDPs.

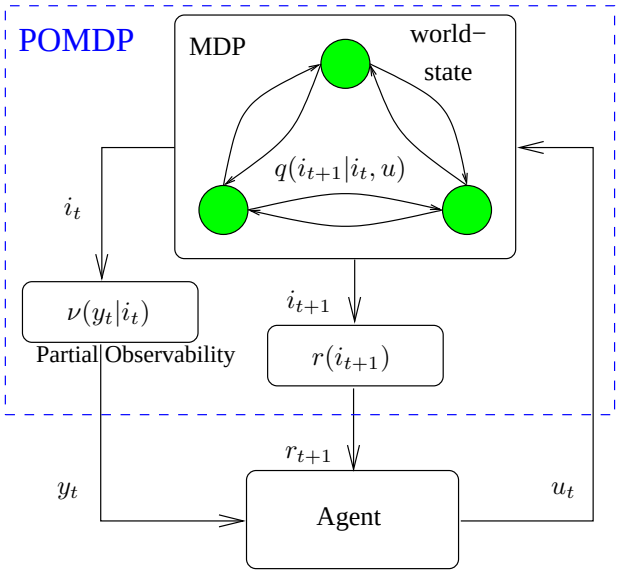


Figure 1: Diagram of the world perspective of POMDP showing the underlying MDP, and the stochastic process $\nu(y_t|i_t)$ mapping the current state i_t to an observation y_t, thus hiding the true state information.

2 Modelling the World as a POMDP

Our setting is that of an agent taking actions in a world according to its policy. The agent receives feedback about its performance through a scalar reward r_t received at each time step.

Definition 1. *Formally, a POMDP consists of:*

- $|\mathcal{S}|$ *states* $\mathcal{S} = \{1, \ldots, |\mathcal{S}|\}$ *of the world;*
- $|\mathcal{A}|$ *actions (or controls)* $\mathcal{U} = \{1, \ldots, |\mathcal{A}|\}$ *available to the policy;*
- $|\mathcal{Y}|$ *observations* $\mathcal{Y} = \{1, \ldots, |\mathcal{Y}|\}$*;*
- *a (possibly stochastic) reward* $r(i) \in \mathbb{R}$ *for each state* $i \in \mathcal{S}$*.*

For ease of exposition we assume $\mathcal{S}, \mathcal{Y}$, and $\mathcal{U}$ are finite. Generalisations to uncountably infinite cases are possible for many algorithms, however, the mathematics is considerably more complex and unlikely to deepen understanding of the underlying algorithms.

Each action $u \in \mathcal{U}$ determines a stochastic matrix $[q(j|i,u)]_{i=1\ldots|\mathcal{S}|, j=1\ldots|\mathcal{S}|}$, where $q(j|i,u)$ denotes the probability of making a transition from state $i \in \mathcal{S}$ to state $j \in \mathcal{S}$ given action $u \in \mathcal{U}$. For each state i, an observation $y \in \mathcal{Y}$ is generated independently with probability $\nu(y|i)$. The distributions $q(j|i,u)$ and $\nu(y|i)$, along with a description of the rewards, constitutes the *model* of the POMDP shown in Figure 1.

Conceptually, the world issues rewards after each state transition. Rewards may depend on the action and the previous state $r_{t+1} = r(i_t, u_t)$; but without

loss of generality we will assume rewards are a function of only the updated state $r_{t+1} = r(j) = r(i_{t+1})$ so that r_1 is the first reward, received after the first action.[1]

If the state is not hidden, that is, there is an observation y for each world state and $\nu(y|i) = 1$ if $y = i$, then we are in the MDP setting. This setting is significantly easier than the POMDP setting with algorithms running in time that is $O(|\mathcal{S}|^2)$ per iteration, or $O(|\mathcal{S}|^3)$ for a closed form solution [Bellman, 1957]. MDP research was the precursor to the study of POMDPs so in Section 5 we briefly mention some of the most influential MDP algorithms.

POMDPs may be *episodic*, where the task ends upon the agent entering a state i^* which is in the set of terminal states $\mathcal{S}^* \subset \mathcal{S}$. More generally, they may be infinite-horizon POMDPs that conceptually run forever.

3 Agents with Internal State

Agents generally require internal state, or memory, to act well. To make this concept concrete, consider the Load/Unload scenario [Peshkin et al., 1999] shown in Figure 2(a). The observations alone do not allow the agent to determine if it should move left or right while it occupies the intermediate `N` observation states. However, if the agent remembers whether it last visited the load or unload location (1 bit of memory) then it can act optimally.

We now introduce a generic internal-state agent model that covers all existing algorithms. The agent has access to a set of internal states $g \in \mathcal{G} = \{1, \ldots, |\mathcal{G}|\}$ (I-states for short). Finite memory algorithms have finite $\mathcal{G}$. Alternatively, exact infinite-horizon algorithms usually assume infinite $\mathcal{G}$.[2] For example, one uncountable form of $\mathcal{G}$ is the set of all *belief states*: distributions over world state occupancy in the $|\mathcal{S}|$ dimension simplex. The cross product of the world-state space $\mathcal{S}$ and the internal-state space $\mathcal{G}$ form the *global*-state space.

The agent implements a *parameterised* policy μ that maps observations $y \in \mathcal{Y}$, and I-states $h \in \mathcal{G}$, into probability distributions over the controls $\mathcal{U}$. We denote the probability under μ of control u, given I-state h, observation y, and parameters $\theta \in \mathbb{R}^{n_\theta}$, by $\mu(u|\theta, h, y)$. Deterministic policies emit distributions that assign probability 1 to action u_t and 0 to all other actions. Policies are learnt by searching the space of parameters $\theta \in \mathbb{R}^{n_\theta}$.

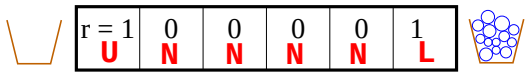
The I-state evolves as a function of the current observation $y \in \mathcal{Y}$, and I-state $g \in \mathcal{G}$. Specifically, we assume the existence of a *parameterised* function ω such that the probability of making a transition to I-state $h \in \mathcal{G}$, given current I-state $g \in \mathcal{G}$, observation $y \in \mathcal{Y}$, and parameters $\phi \in \mathbb{R}^{n_\phi}$, is denoted by $\omega(h|\phi, g, y)$. The I-state transition function may be learnt by searching the space of parameters $\phi \in \mathbb{R}^{n_\phi}$.

An important feature of our model is that both the policy and the I-state transitions can be stochastic.

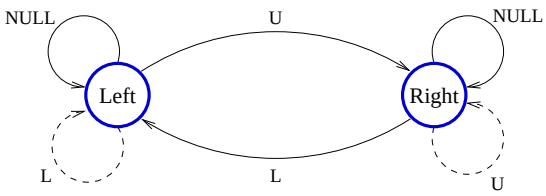
Some algorithms do not learn the I-state transition function, fixing ϕ in advance. Examples include the exact methods of Section 6.1, where we can write

[1] The more general case can be obtained by augmenting the state with information about the last action (and optionally the last state), implicitly making the reward a function of the last action.

[2] It is an abuse of notation to write $g \in \mathcal{G} = \{1, \ldots, |\mathcal{G}|\}$ when $\mathcal{G}$ could be uncountably infinite, however, we ignore this since it is immaterial to the discussion. Most of the discussed algorithms assume $\mathcal{G}$ is finite.



(a)



(b)

Figure 2: 2(a) The Load/Unload scenario with 6 locations. The agent receives a reward of 1 each time it passes through the unload location `U`, or the load location`L`, after having first passed through the opposite end of the road. In each state the agent has a choice of two actions: either `left` or `right`. The three observations are $\mathcal{Y} = \{$`U, N, L`$\}$, which are issued in the unloading state, the intermediate states and the loading states respectively. 2(b) The policy graph learned for the Load/Unload scenario. Each node represents an internal state. The "Left" state is interpreted as: *I have a load so move left,* and the "Right" state as: *I dropped my load so move right.* The dashed transitions are used during learning but not by the final policy.

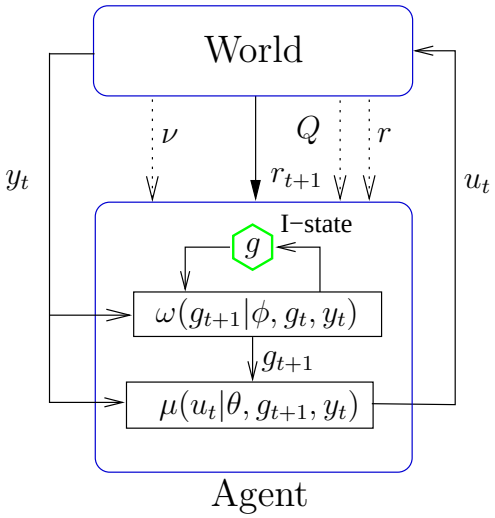


Figure 3: POMDP from the agent's point of view. The dashed lines represent the extra information available for model-based algorithms.

down the equations that determine the next belief state from the previous belief state and current observation. Throughout this section we compare algorithms in terms of how the $\omega(h|\phi, g, y)$ and $\mu(u|\theta, h, y)$ functions work and how they are parameterised. Tables 1 and 2 summarise these differences at the end of the paper.

The agent's view of the POMDP is represented by Figure 3. The POMDP evolves as follows:

1. let $i_0 \in \mathcal{S}$ and $g_0 \in \mathcal{G}$ denote the initial state of the world and the initial I-state of the agent respectively;
2. at time step t, generate an observation $y_t \in \mathcal{Y}$ with probability $\nu(y_t|i_t)$;
3. generate a new I-state g_{t+1} with probability $\omega(g_{t+1}|\phi, g_t, y_t)$;
4. generate action u_t with probability $\mu(u_t|\theta, g_{t+1}, y_t)$;
5. generate a new world state i_{t+1} with probability $q(i_{t+1}|i_t, u_t)$;
6. $t = t + 1$, goto 2.

4 Long-Term Rewards

POMDP algorithms attempt to globally or locally optimise $\theta \in \mathbb{R}^{n_\theta}$ and $\phi \in \mathbb{R}^{n_\phi}$, maximising the *long-term average reward*:

$$\eta(\phi, \theta, i, g) := \lim_{T \to \infty} \frac{1}{T} \mathbb{E}_{\phi,\theta} \left[\sum_{t=0}^{T} r(i_t) | i_0 = i, g_0 = g \right], \tag{1}$$

where $\mathbb{E}_{\phi,\theta}$ denotes the expectation over all global trajectories $\{(i_0, g_0), \ldots, (i_T, g_T)\}$ when the agent is parameterised by ϕ and θ. An alternative measure of performance is the *discounted sum of rewards*, introducing a discount factor $\beta \in [0, 1)$

$$J_\beta(\phi, \theta, i, g) := \mathbb{E}_{\phi,\theta}\left[\sum_{t=0}^{\infty} \beta^t r(i_t) | i_0 = i, g_0 = g\right]. \tag{2}$$

The exponential decay of the impact of past rewards is equivalent to the assumption that actions have exponentially decaying impact on the current performance of the agent as time goes on. Alternatively, this can be viewed as the assumption that rewards are exponentially more likely to be generated by the most recent actions.

We prefer the long-term average because it gives equal value to all rewards received throughout the evolution of the POMDP. However, the discounted version is useful because it allows the solution of the *temporal credit assignment* problem in infinite-horizon POMDPs by enforcing an effective horizon [Sutton and Barto, 1998]. Fortunately, when the agent is suitably *mixing*, maximising one is equivalent to maximising the other. Suitably mixing means that $\eta(\phi, \theta, i, g)$ is independent of the starting state (i, g) and so may be written $\eta(\phi, \theta)$. In other words, the POMDP is *ergodic*. In this case, there is also a unique stationary distribution over world/internal state pairs. We denote the expectation over this stationary distribution with $\mathbb{E}_{i,g}$. For ergodic POMDPs the long-term average reward and the discounted reward are related by [Baxter and Bartlett, 2001]

$$\mathbb{E}_{i,g} J_\beta(\theta, \phi, i, g) = \frac{1}{1-\beta} \mathbb{E}_{i,g} \eta(\phi, \theta, i, g).$$

5 Methods for Solving MDPs

In this section we describe dynamic programming methods that apply when the state is fully observable, that is, the MDP setting. We will restrict ourselves to the discounted reward setting though the methods can often also be applied to average rewards.

If we have a method of determining the long-term *value* of each state then the agent can act by choosing the action that leads to the state with the highest value. The value is the expected discounted reward for entering state i under the optimal policy. Bellman [1957] describes a procedure known as dynamic programming (DP) which allows us to determine the value $J^*_\beta(i)$ for each state $i \in \mathcal{S}$. We drop the dependency of $J^*_\beta(i)$ on ϕ and θ for this section because DP computes the optimal value directly rather than the value gained by a particular agent. The MDP assumption means that memory is not necessary. This is equivalent to having a single internal state, making the memory process $\omega(h|\phi, g, y)$ trivial. Such agents are said to implement *reactive* policies. DP is described by the *Bellman equation*, where $\beta \in [0, 1)$ is a discount factor that weights the importance of the instantaneous reward against the long-term reward

$$J^*_\beta(i) = \max_u \left[r(i) + \beta \sum_{j \in \mathcal{S}} q(j|i, u) J^*_\beta(j) \right]. \tag{3}$$

Assuming we can store a value estimate for each state, DP proceeds by replacing $J^*_\beta(i)$ with an estimate $J_\beta(i)$ and iterating

$$J_\beta(i) \leftarrow \max_u \left[r(i) + \beta \sum_{j \in \mathcal{S}} q(j|i,u) J_\beta(j) \right]. \tag{4}$$

In the limit as the number of iterations goes to infinity, $J_\beta(i)$ converges to $J^*_\beta(i)$ [Bertsekas and Tsitsiklis, 1996]. Once $J^*_\beta(i)$ is known the optimal policy is given by

$$u^*_i = \arg\max_u \left[r(i) + \beta \sum_{j \in \mathcal{S}} q(j|i,u) J^*_\beta(j) \right].$$

To reflect full observability, and the absence of internal state, we write the policy as $\mu(u|\theta, i)$ instead of $\mu(u|\theta, h, y)$. For MDPs the optimal μ is deterministic and equal to

$$\mu(u|\theta, i) = \chi_u(u^*_i), \tag{5}$$

where $\chi_m(k)$ is the indicator function

$$\chi_m(k) := \begin{cases} 1 & \text{if } k = m, \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

Recall that the vector θ parameterises the policy. In the MDP case θ could represent the value of each state $i \in \mathcal{S}$ or it could represent the mapping of states to actions derived from those values. In the later case θ would be a vector of length $|\mathcal{S}|$, representing a table mapping states directly to the optimal action for that state. When the state and action spaces are large we resort to some form of function approximator to represent the table. For example, the parameters θ could be the weights of an artificial neural network (ANN) that maps states to actions. Function approximation for DP was in use by Bellman et al. [1963] and possibly earlier.

Iterating Equation (4) until convergence, and forming a policy from Equation (5) is the basis of *value iteration* [Howard, 1960]. Alternatively, *policy iteration* [Bellman, 1957] learns the value of states under a particular agent, denoted $J_{\beta,\mu}(i)$. Once the value of the policy has been learnt the policy is updated to maximise $J_{\beta,\mu}(i)$, followed by a re-evaluation of $J_{\beta,\mu}(i)$ under the new policy. This is repeated until the policy does not change during maximisation. Policy iteration resembles Expectation-Maximisation (EM) methods [Dempster et al., 1977] because we estimate the expected value for state i, then alter the policy to maximise the expected value for i.

Evaluating (4) has complexity $O(|\mathcal{A}||\mathcal{S}|^2)$ per step which, while polynomial, is infeasible for very large state spaces. Also, the transition probabilities $q(j|i,u)$, which are part of the model, may not always be available. These two observations motivate Monte-Carlo methods for computing $J^*_\beta(i)$. These methods learn by interacting with the world and gathering experience about the long-term rewards from each state. Q-learning is one algorithm for learning value function $Q(i,u) : \mathcal{S} \times \mathcal{U} \mapsto \mathbb{R}$, which represents the value of taking action u in state i and then acting optimally. It is summarised by the following update rule [Sutton et al., 1999] that introduces a step size $\gamma_t \in [0, 1)$

$$Q(i_t, u_t) \leftarrow Q(i_t, u_t) + \gamma_t [r_{t+1} + \beta \max_{u'} Q(i_{t+1}, u') - Q(i_t, u_t)],$$

which states that the value $Q(i_t, u_t)$ should be updated in the direction of the error $[r_{t+1} + \beta \max_{u'} Q(i_{t+1}, u')] - Q(i_t, u_t)$. The values converge provided an independent value is stored for each state/action pair, each state is visited infinitely often, and γ is decreased in such a way that $\sum_t \gamma_t = \infty$ and $\sum_t \gamma_t^2 < \infty$ [Mitchell, 1997]. Q-learning is an *off-policy* method, meaning the optimal policy can be learnt by following a fixed alternative policy. This allows a reduction of the number of expensive world interactions by repeatedly re-using experience gathered under an old policy to improve the current policy.

The complexity of learning $J^*_\beta(i)$ or $Q(i, u)$ can be reduced by automatically aggregating states into clusters of states, or meta-states, at the cost of introducing partial observability [Singh et al., 1995, Engel and Mannor, 2001a].

Readers interested in MDPs are referred to books such as Puterman [1994], which covers model-based algorithms and many variants on MDPs. Bertsekas and Tsitsiklis [1996] provides an analysis of the convergence properties of MDP algorithms and Kaelbling et al. [1996] describes the algorithms from the *reinforcement learning* point of view.

Formalisms that sit between POMDPs and MDPs can simplify policy generation and analysis. For example, hPOMDPs require that remembering the history of observations/actions pairs $h_T = \{(y_0, u_0), (y_1, u_1), \ldots, (y_T, u_T)\}$ will allow the agent to determine the true state; which is useful when analysing MDP algorithms under limited partial observability [Pendrith and McGarity, 1998]. Even-odd POMDPs assume the world is fully observable at every other time step, limiting the implications of partial observability [Zubeck and Dietterich, 2000]. However, we shall focus on the more general class of POMDPs for the rest of this paper.

6 Learning with a Model

This section describes existing methods for producing POMDP agents when the model of the POMDP is known, which is equivalent to knowing $\nu(y|i)$, $q(j|i, u)$ and $r(i)$. These include *exact methods* that are guaranteed to learn the optimal policy given sufficient, possibly infinite, time and memory. We define the optimal agent as the agent that obtains the maximum possible long-term (average or discounted) reward given that the agent does not have access to the true state of the system. The long-term reward of optimal MDP methods upper bounds the reward that can be obtained after introducing partial observability [Hauskrecht, 2000].

6.1 Exact Methods

The observation process may not reveal sufficient state to allow reactive policies to choose the optimal action. One solution is to model an agent that remembers its entire observation/action history $\bar{y}$. This fits within our framework by setting $\mathcal{G}$ to be the sequence of all past observations and actions $\bar{y} = \{(y_0, u_0), \ldots, (y_t, u_t)\}$. Using $\bar{y}$ may allow the agent to determine the true state and hence act optimally. Even if $\bar{y}$ does not allow the agent to determine the true state, it may help to reduce the entropy of the agent's belief of which state it occupies, consequently improving the probability that the agent will act correctly. In this case $\mathcal{G}$ is the possibly infinite set of all observation/action

trajectories and $\omega(h|\phi, g, y)$ is a deterministic function that simply concatenates the last observed (y_t, u_t) to $\bar{y}$.

Consider an agent in a symmetric building. If it only receives observations about what is in its line of sight, then many places in the building will appear identical. However, if it remembers the last time it saw a landmark, such as the front doors of the building, then it can infer where it is in the building from its memory of how it moved since seeing the landmark. This is the approach taken by methods such as utile distinction trees [McCallum, 1996] and prediction suffix trees [Ron et al., 1994]. These methods do not necessarily assume knowledge of the POMDP model and will be discussed in Section 7.

6.1.1 Belief States

Explicitly storing $\bar{y}$ results in inefficient memory use because we potentially need to store an infinite amount of history in infinite-horizon settings. Åström [1965] described an alternative to storing histories which is to track the *belief state*: the probability distribution over world-states, $\mathcal{S}$, given the observation and action history $\bar{y}$. The belief state is a sufficient statistic in the sense that the agent can perform as well as if it had access to $\bar{y}$ [Smallwood and Sondik, 1973]. We use the notation $b_t(i|\bar{y}_t)$ to mean the probability that the world is in state i at time t given the history up to time t. Given a belief state b_t, an action u_t, and an observation y_t, the successor belief state is computed using

$$b_{t+1}(j|\bar{y}_t) = \frac{\nu(y_t|j)\sum_{i\in\mathcal{S}} b_t(i|\bar{y}_t)q(j|i,u_t)}{\sum_{y'\in\mathcal{Y}}\nu(y'|j)\sum_{i\in\mathcal{S}} b_t(i|\bar{y}_t)q(j|i,u_t)}. \tag{7}$$

In this setting $\mathcal{G}$ is the possibly uncountable set of belief states the system can reach. From this point on we will use $\mathcal{B}$ instead of $\mathcal{G}$ to refer to the set of reachable belief states, allowing us to keep the belief state distinct from other forms of internal state such as finite state controller I-states. Each element $b \in \mathcal{B}$ is a *vector* in an $|\mathcal{S}|$ dimensional simplex. The function $\omega(b_{t+1}|\phi, b_t, y)$ is deterministic, giving probability 1 to vector b_{t+1} defined by Equation (7).

The set $\mathcal{B}$ defines the states of an equivalent MDP on which DP can be performed by replacing the states in Equation (3) with belief states [Smallwood and Sondik, 1973, Cassandra et al., 1994]

$$\bar{J}_\beta(b') \leftarrow \max_u \left[\bar{r}(b') + \beta\sum_{b\in\mathcal{B}}\bar{q}(b|b',u)\bar{J}_\beta(b)\right], \tag{8}$$

which converges in finite time to within ϵ of the optimal policy value [Lovejoy, 1991]. The bar on $\bar{J}_\beta(b)$ indicates that we are learning values of belief states b instead of world states i. The simplicity of this equation is misleading because $\bar{J}_\beta$, $\bar{r}$, and $\bar{q}$ are represented in terms of their MDP counterparts in Equation (3). For example, $\bar{r}(b) = \sum_{i\in\mathcal{S}} b_i r(i)$. When the set of reachable belief states can be infinite, $\bar{J}_\beta(b)$ is significantly more complex than $J_\beta(i)$, which is the subject of the next section.

6.1.2 Value Function Representation

Maintaining and updating independent values for infinitely many belief states is infeasible. Fortunately, it has been shown for finite horizons that the value over

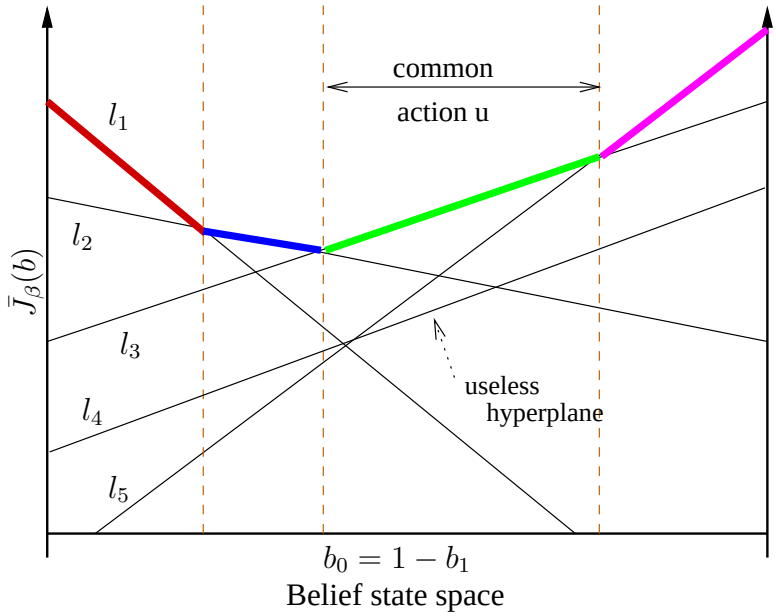


Figure 4: Convex piecewise linear representation of a value function for a continuous belief state with $|\mathcal{S}| = 2$. The plot is a slice from a 3-dimensional plot defined by valid points in the the 2-dimensional simplex, that is, where $b_0 = 1 - b_1$. If the set $\mathcal{L}$ is parsimonious then each l_n defines a line segment which is maximum in some partition of belief space. Each such partition has a unique optimal action u_n^*.

all belief states can be represented exactly by a convex piecewise linear function [Smallwood and Sondik, 1973]. Such a representation is shown for a 2 state system in Figure 4. The set $\mathcal{L}$ contains the hyperplanes needed to represent the value function. Each $l \in \mathcal{L}$ is an $|\mathcal{S}|$ dimensional vector such that the value of hyperplane l for belief state b is $b \cdot l$. The value of $\bar{J}_\beta(b)$ is the maximum over all hyperplanes

$$\bar{J}_\beta(b) = \max_{l \in \mathcal{L}} \{b \cdot l\}.$$

To be useful, a hyperplane l must be the maximum for some b. If $\mathcal{L}$ contains only useful hyperplanes then it is called a *parsimonious* set [Zhang, 1995].

Estimating the value function proceeds by using value iteration on belief states chosen to be points in the belief simplex that have a unique maximuml l, and that have not yet converged to their true value. These points, as found by the Witness algorithm, are called *witness points* [Cassandra et al., 1994]. This term is now often used to describe any belief state upon which it is useful to perform a DP update. Such updates generate new vectors to be added to $\mathcal{L}$. A difficult task — and the way in which most exact POMDP algorithms differ — is determining the witness points efficiently. This is often done by solving a set of linear programs. Once a single round of value iteration has been performed $\mathcal{L}$ is examined to remove useless vectors, creating a parsimonious set ready for the next round.

The simplest exact method is Sondik/Monahan's enumeration algorithm [Monahan, 1982], which enumerates all hyperplanes in $\mathcal{L}$ and performs a DP update on each one. This method results in a large number of new vectors. In rough order of increasing sophistication other algorithms include the One-Pass algorithm of Sondik [1971], the Linear Support algorithm of Cheng [1988], the Witness algorithm of Cassandra et al. [1994], and the Incremental Pruning algorithm of Zhang and Liu [1996]. Cassandra [1999] provides a non-mathematical comparison of these algorithms.

6.1.3 Policy Representation

POMDP agents can be represented by a policy graph that is a directed and possibly cyclic graph. Each node is labelled with a single action and transitions out of each node are labelled with observations. All nodes map to a polyhedral partition in the value function. Partitions are defined by the region where a single hyperplane $l \in \mathcal{L}$ maximises $\bar{J}_\beta(b)$ [Cassandra et al., 1994]. Transitions between nodes in the policy graph are a deterministic function of the current observation. Actions are a deterministic function of the current node. If the optimal policy can be represented by a cyclic graph with a finite number of nodes, then the POMDP is called *finitely transient* [Sondik, 1978].

A policy can be represented by $\mu(u|\theta, h, y)$ in several ways. The parameters θ could be used to store hyperplanes in $\mathcal{L}$, in which case $\mu(u|\theta, b_{t+1}, y_t)$ gives probability 1 to the action associated with the hyperplane that maximises the value at b_t. If the policy is finitely transient it can be represented more compactly by the policy graph. In this case the internal states $\mathcal{G}$ are equivalent to the policy-graph nodes. The I-state (internal state) update $\omega(h|\phi, g, y)$ uses y to index the correct transition from node g to node h. The policy $\mu(u|\theta, h)$ is simply a lookup table that gives the optimal action for each node h. Thus, ϕ represents policy-graph transitions, and θ maps policy-graph nodes to actions.

In many cases it is possible for large infinite-horizon POMDPs to be controlled well by simple policy graphs. Consider the Load/Unload scenario [Peshkin et al., 1999] shown in Figure 2(a). The optimal policy graph is shown in Figure 2(b). This policy graph suffices no matter how many intermediate locations there are between the load and unload locations. As the number of intermediate locations increases, the value function becomes more complex but the optimal policy graph does not. This example partially motivates the idea (discussed in Section 6.7) of searching in the space of policy graphs instead of learning value functions.

6.1.4 Complexity of Exact Methods

There are two problems with exact methods that make them intractable for problems with more than a few tens of states, observations, and actions. To discuss the first we introduce the concept of *state-variables*. State variables describe the state in terms of features that are true or false, which is an arguably more intuitive description than enumerating states. Consider a system with v boolean state variables. The number of POMDP states is $|\mathcal{S}| = 2^v$, thus $|\mathcal{S}|$ grows exponentially with the number of state variables. For example, two state variables might be "is it raining?" and "is the umbrella open?" requiring 4 states to encode.

Since DP for POMDPs involves updating belief states, the complexity of POMDP algorithms grows exponentially with the number of state variables. This makes belief-state monitoring infeasible for large problems [Boyen and Koller, 1998, Sallans, 2000].

The second problem is representing $\bar{J}_t(b)$, the value function after t steps of DP. Let $\mathcal{B}_t$ be the set of belief states reachable at time t. Recall that $|\mathcal{Y}|$ is the number of possible observations. Assuming $|\mathcal{B}_0| = 1$, that is, a single known initial belief state, then after t steps of a greedy policy we potentially have $|\mathcal{Y}|^t$ belief states in $\mathcal{B}_t$. Thus, the problem of representing $\bar{J}_\beta(b)$ grows exponentially in the horizon length. Since the belief-state space is infinite for infinite horizons, exact methods perform DP on the hyperplanes in $\mathcal{L}$. This representation grows exponentially in the observations since a single DP step in the worst case results in $|\mathcal{L}_{t+1}| = |\mathcal{A}||\mathcal{L}_t|^{|\mathcal{Y}|}$ [Cassandra, 1998]. Furthermore, evaluating if there exists a belief state b for which an $l \in \mathcal{L}_{t+1}$ is dominant requires solving expensive linear programs.

Even for simplified *finite*-horizon POMDPs, the problem of finding the optimal policy is PSPACE-hard [Papadimitriou and Tsitsiklis, 1987]. Learning a policy graph with a *constrained* number of nodes is NP-hard [Meuleau et al., 1999a]. Infinite-horizon POMDPs can result in an infinite number of belief states or hyperplanes l, resulting in the problem of determining convergence being undecidable [Madani et al., 1999]. Even worse, determining ϵ-convergence is undecidable in polynomial time for *finite*-horizon POMDPs [Lusena et al., 2001]. Despite this gloomy theoretical message, empirical results show that it is possible to learn reasonable policies in reasonable time for POMDPs of ever increasing complexity. We avoid the intractable computational complexity of exact methods by abandoning the requirement that policies be optimal, while retaining the requirement that agents should at least converge to a fixed policy.

6.2 Approximate Value Function Methods

This section introduces model-based methods that learn approximations to $J^*_\beta(i)$. For a more detailed survey of these methods see Hauskrecht [2000].

6.2.1 Heuristics for Exact Methods

A number of methods simplify the representation of $\bar{J}_\beta(b)$ by assuming the system is an MDP and learning the underlying Q-function $Q(i, u)$. This must be done via model-based methods or by computer simulation because the partial observability of the real world does not allow i to be known during real-world interaction.

One choice for the policy is [Nourbakhsh et al., 1995]

$$\bar{u}^*(b) = \arg\max_u Q(\arg\max_j b(j), u), \qquad \mu(u|\theta, b) = \chi_u(\bar{u}^*(b)),$$

which assumes the agent is in the most likely state (MLS), known as the MLS heuristic. This approach completely ignores the agent's confusion about which state it is in. The voting heuristic [Simmons and Koenig, 1995] weights the vote for the best action in each state by the probability of being in that state

$$u^*(j) = \arg\max_a Q(j, a), \qquad \bar{u}^*(b) = \arg\max_u \sum_{j \in \mathcal{S}} b(j)\chi_u(u^*(j))Q(j, u^*(j)).$$

The popular Q_{MDP} heuristic [Littman et al., 1995] takes into account the belief state for one step and then assumes that the state is entirely known [Cassandra, 1998]

$$\bar{u}^*(b) = \arg\max_u \sum_{j \in \mathcal{S}} b(j) Q(j, u).$$

These heuristics will perform poorly if the belief state is close to uniform. Due to the convexity of the value function, $\bar{J}_\beta(b)$ generally grows as the entropy of b decreases. The highest expected payoffs occur at the simplex corners. This motivates choosing actions that decrease the entropy of the belief state in the hope that the heuristics above will perform better with a peaked belief. For example, consider a robot that must reach the other side of a featureless desert [Roy and Thrun, 2001]. If it goes straight across it will quickly become lost due to lack of landmarks and movement errors. The better policy is to skirt the desert, taking longer but remaining certain of reaching the goal because the robot is certain of its location. Cassandra [1998] shows how to use the entropy of b to switch between *information gathering* policies and exploitive policies. The entropy can also be used to weight two policies that trade off information gathering and exploitation. These heuristics may be misleading if the minimum of $\bar{J}_\beta(b)$ does not occur at the point of greatest entropy, that is, the uniform belief state.

An alternative family of heuristics are based on simplified versions of the full DP update. For example, determining maximising vectors in $\mathcal{L}$ for each state, rather than over all states, produces the *fast informed bound* of Hauskrecht [1997]. The heuristics in this section are also useful as upper bounds on J^*_β, allowing them to direct tree search procedures used in classical planning approaches (see Section 6.4).

6.2.2 Grid Methods

Value functions over a continuous belief space can be approximated by values at a finite set of points along with an interpolation rule. Once a set of grid points has been chosen an equivalent MDP can be constructed where the states are the grid points. This POMDP can be solved in polynomial time [Hauskrecht, 2000]. The idea is equivalent to constructing a policy graph where each node is chosen heuristically to represent what might be an "interesting" region of belief state space. The two significant issues are how to choose grid points and how to perform interpolation.

Regular grids are an obvious choice but they fail to scale for large state spaces [Lovejoy, 1991]. Methods for choosing irregular grids include the use of simulation to determine useful grid points [Hauskrecht, 1997] and adding points where large variations in values are detected for two local points that have observations in common [Brafman, 1997].[3]

Interpolation schemes should maintain the convex nature of the value function and hence are typically of the form

$$\bar{J}_\beta(b) = \arg\max_u \sum_{g \in \mathcal{G}} \lambda_{g,b} f(g, u), \tag{9}$$

[3]The mutual observations requirement is an interesting heuristic that may identify whether belief states between the two existing points can be reached.

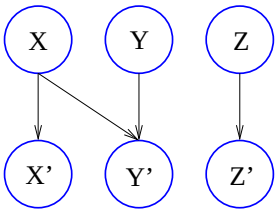


Figure 5: Two-slice temporal Bayes network showing dependencies between state variables over successive time steps. X and Z depend only on themselves but Y is influenced by itself and X. This conditional dependence structure can be exploited to efficiently model the transition probabilities of a POMDP.

where $\sum_{g \in \mathcal{G}} \lambda_{g,b} = 1$ and $\lambda_{g,b} \geq 0 \ \forall g, b$. The function $f(g, u)$ represents the value grid point g under action u. Examples include nearest neighbour, linear interpolation, and kernel regression [Hauskrecht, 2000]. The MLS heuristic can be thought of as a grid method with points at the belief-simplex corners and a simple 1-nearest-neighbour interpolation (assuming an L_∞ distance measure) [Brafman, 1997].

Recently, ϵ-convergence was shown for a grid method by using ϵ-covers under a specifically chosen distance metric [Bonet, 2002]. Within our knowledge it is the only provably optimal grid based algorithm. It is still intractable because the number of grid points required grows exponentially with the size of the state space.

Alternatively, grid point value estimation steps can be interleaved with exact hyperplane DP updates in order to speed up DP convergence without sacrificing policy optimality [Zhang and Zhang, 2001].

In the context of grid methods the $\omega(b_{t+1}|\phi, b_t, y_t)$ process still performs a deterministic update on the belief state, but now $\mu(u|\theta, b_{t+1}, y_t)$ represents the choice of action based on the interpolated value from (9). The θ parameters store the values of actions at the grid points.

6.3 Factored Belief States

Using the state variable description of a POMDP (discussed in Section 6.1.4) is sometimes referred to as belief factorisation, especially when the factorisation is not exact. The value of a state variable X_t may depend (approximately) on a small subset of the state variables at the previous time steps. Transition probabilities can be represented by a two-slice temporal Bayes net (BN) that models the dependencies between state variables over successive time steps as a 2 layer acyclic graph [Dean and Kanazawa, 1989] (see Figure 5). Each node contains a conditional probability table showing how its parents affect the probability of the state variable being true.

Alternatively, the state variable dependencies can be represented by a tree structure such as an *algebraic decision diagrams* (ADD) [Bahar et al., 1993]. BNs and ADDs are applied to POMDPs by Boutilier and Poole [1996] to simplify both the belief monitoring problem and the value function representation problem. We now describe how recent work has used belief factorisation to solve each of these problems.

6.3.1 Efficiently Monitoring Belief States

Boyen and Koller [1998] observed that representing belief states as BNs can lead to an accumulation of errors over many time steps that result in the belief state approximation diverging. The authors show that projections of the BN that produce strictly independent groups of state variables results in converging belief-states, allowing recovery from errors. These projections can be searched to automatically determine BNs that perform well under a specified reward criterion. Heuristic methods to determine factorisations were introduced by Poupart and Boutilier [2000]. These were improved in Poupart and Boutilier [2001] which presents a vector-space analysis of belief-state approximation, providing a formally motivated search procedure for determining belief-state projections with bounded errors. This expensive off-line calculation speeds up on-line belief state tracking.

An alternative method of searching for the optimal belief factorisation is to learn a dynamic sigmoid belief network [Sallans, 2000]. Stochastic gradient ascent is performed in the space of BNs with a fixed number of dependencies, minimising the error between the belief state and the approximation. This algorithm has been applied to the large New York driving problem of McCallum [1996]. Unlike the algorithms considered so far, which apply a fixed I-state update through $\omega(h|\phi, g, y)$, this algorithm can be viewed as learning $\omega(h|\phi, g, y)$ by adjusting the belief network parameterised by ϕ.

6.3.2 Factored Value Functions

So far we have discussed piecewise linear value functions represented by sets of hyperplanes $\mathcal{L}$. Koller and Parr [2000] note that even if compact BN representations of the transition model are found, they may not induce a similar structure on the value function. The same paper discusses factored value functions implemented as weighted linear combinations of polynomial basis functions. The weights are chosen to minimise the squared error from the true value. In our model of agents the weights would be represented by θ.

This idea is combined with BNs for belief monitoring by Guestrin et al. [2001]. With approximate belief monitoring and factored linear value functions, DP methods become more feasible since learning reduces to solving systems of linear equations instead of a linear program for every vector in $\mathcal{L}$. The empirical advantages of factored value functions are studied in [Hansen and Feng, 2000] which uses algebraic decision diagrams to represent $\bar{J}_\beta(b)$.

6.4 Classical Planning

The link between classical planning algorithms such as C-Buridan [Draper et al., 1994] and POMDP algorithms is summarised by Blythe [1999]. One planning approach is to search the tree of possible future histories $\bar{y}$ to find the root action with the highest expected payoff. Branch and bound approaches such as the AO^* algorithm [Nilsson, 1980, §3.2] can be used, searching the tree to a finite depth and pruning the search by eliminating branches that have upper bounds below the best current lower bound. At the leaves the upper and lower bounds can be estimated using Q_{MDP} and the value-*minimising*-action respectively [Washington, 1997].

This idea is applied to *factored* finite-horizon POMDPs by McAllester and Singh [1999]. While avoiding the piecewise linear value function, the algorithm is still exponential in the horizon due to the tree search procedure. Furthermore, since there is no concise policy representation the exponential complexity step is performed during normal policy execution instead of only during a learning phase. In this case $\omega(b_{t+1}|\phi, b_t, y_t)$ is responsible for deterministically tracking the factored belief state, and $\mu(u|\theta, b)$ is responsible for performing the tree search in order to find the best u. The parameters θ are not learnt, but ϕ could be learnt to optimise factored belief state tracking, as in Section 6.3.1.

6.5 Simulation and Belief States

The full DP update of Equation (8) is inefficient because all states are assumed equally important. Simulation algorithms learn value functions from real experience, concentrating learning effort on the states most likely to be encountered. This idea can be applied to learning Q-functions for finite-horizon POMDPs by simulating a path through the POMDP while monitoring the current belief state and performing an iteration of (8) for the current point in belief state space instead of the associated $l \in \mathcal{L}$ [Geffner and Bonet, 1998]. This algorithm, called RTDP-Bel, does not address the problem of representing $Q(b, u)$ for many belief states over large horizons. However, it has been applied to solving finite-horizon POMDPs with hundreds of states and long-term memory.

Usually we need to learn Q-functions that generalise to all b. A simple approach is Linear-Q which defines $Q(b, u) = \phi_u \cdot b$ [Littman et al., 1995]. This method is equivalent to training a linear controller using stochastic gradient ascent with training patterns given by inputs b_t and target outputs $r + \beta \max_u Q(b_{t+1}, u)$. The SPOVA algorithm [Parr and Russell, 1995] is essentially the same scheme, using simulation to generate gradients for updating the smooth approximate function

$$\bar{J}_\beta(b) = \left[\sum_{l \in \mathcal{L}} (b \cdot l)^k \right]^{\frac{1}{k}},$$

where k is a tunable smoothing parameter and the number of hyperplanes $|\mathcal{L}|$ is fixed. Artificial neural networks (ANNs) can also be used to approximate value functions with either the full belief state or a factored representation used as the input [Rodríguez et al., 2000]. Both Linear-Q and SPOVA learn the θ parameters but not ϕ.

6.6 Continuous State and Action Spaces

Thrun [2000] extends value iteration for belief states into continuous state spaces and action spaces. Since infinite memory would be needed to exactly represent belief states, Thrun uses a sampled representation of belief states. The algorithm balances accuracy with running time by altering the number of samples n used to track the belief state. The belief state is updated using a particle filter [Fearnhead, 1998] that converges to the true belief state as $n \to \infty$, for arbitrary continuous distributions $q(j|i, u)$ and $\nu(y|i)$. Approximate continuous belief states are formed from the samples using Gaussian kernels. The hyperplane value-function representation cannot be used because the belief state is

a continuous distribution, equivalent to infinitely many belief state dimensions. Even function approximators like neural networks cannot trivially contend with continuous belief states. Instead, the value function is approximated using a nearest neighbour method where the output value is the average of the k nearest neighbours (KNN) of previously evaluated infinite-dimension belief states.

Particle filters can also be used to monitor the belief state during policy execution. Poupart et al. [2001] analyses the value loss incurred by using particle filters once a value function has already been learnt. The paper uses this analysis to develop an adaptive scheme for choosing n based on the probability of choosing an approximately optimal action. These methods may also help in finite domains with many states.

6.7 Policy Search

The approaches we have discussed so far have learnt values for each state and action. Values can be explicitly encoded in θ or used to manufacture a more compact policy graph. This section introduces methods that learn policies directly, sometimes avoiding value estimation at all. For example, we can attempt to learn a set of parameters θ that directly encodes a policy graph.

There are several reasons to prefer policy search methods to value function methods, particularly in the presence of partial observability. Informally, Occam's Razor can be invoked in favour of policy search methods by noting that it is intuitively simpler to determine *how to act* instead of the *value of acting*. For example, Section 6.1.3 demonstrated that a 2 node policy graph can optimally control an arbitrarily large POMDP. The value function representation would require storing or approximating a value for each state.

More formally, approximate value function methods can perform poorly in POMDP settings because they usually find deterministic policies (by application of the max function in Equation (8)). Approximate POMDP methods generally require stochastic policies [Singh et al., 1994]. Value methods in function approximation settings also lack convergence guarantees because small changes to state values can cause discontinuous changes to the policy [Bertsekas and Tsitsiklis, 1996].

However, policy search is still difficult. Even for restricted classes of policies, the search is NP-hard [Meuleau et al., 1999a]. Value function methods have the advantage of imposing the useful Bellman constraint. Consequently, value functions are likely to remain useful for small to medium size problems.

Policy search can be implemented using *policy iteration* [Bertsekas and Tsitsiklis, 1996, §2.2.3]. The first step is to evaluate the current policy and the second step is policy improvement. Sondik [1978] described how policy evaluation could be performed by representing the policy as a set of polyhedral partitions in belief space. Instead, Hansen [1998] uses the policy-graph representation, greatly simplifying the task of evaluating the policy. Evaluating the policy is performed by solving a set of linear equations that gives the hyperplane representation of the value function $\bar{J}_\beta(b)$. Policy improvement works by adding or merging policy graph nodes based on how the set of vectors in $\mathcal{L}$ changes after a single hyperplane update on the evaluated policy. Hansen also describes a branch and bound tree search procedure that can be used to approximate the DP step, achieving an order of magnitude speed-up.

The nodes of a policy graph can be interpreted as states of an internal MDP that is observable by the agent. The process defined by the cross product of world states with policy-graph states is Markov [Hansen, 1998]. The value of the cross product MDP can be computed using value iteration. Meuleau et al. [1999a] finds optimal policy graphs within a constrained space using a branch and bound search method, computing the value of complete policies at the leaves only when needed.

The same paper also shows how gradient ascent can be used to search the space of stochastic policy graphs, which we refer to as *finite state controllers* (FSCs). FSCs are described in detail in Section 7.2.5. Given a model, the gradient of the *discounted* sum of rewards can be computed directly, requiring large matrix inversions. This is the first algorithm discussed so far that does not evaluate values explicitly. Because it does this with gradient ascent it is our first example of a *policy-gradient* method. It is also our first example of a stochastic internal-state function.

The simulation methods of Section 6.5 can be used to generate policies instead of value functions. For example, if the belief state is tracked during simulation and used as an input to a neural network, then network outputs can be mapped to a distribution over actions.

6.8 Hybrid Value-Policy Methods

MDP Value methods may fail to converge for POMDPs or when using function approximation in the MDP setting. In contrast, policy-gradient methods converge to a local maximum under mild conditions. However, policy-gradient methods exhibit high variance [Marbach and Tsitsiklis, 2000]. One reason for high variance is that they ignore the Bellman equation which is a useful constraint for agents [Peshkin et al., 1999, Baird and Moore, 1999]. In an attempt to combine the low variance of value based methods with the convergence to local maximum guarantees of policy-gradient methods, Baird and Moore [1999] introduced VAPS. This algorithm specifies a parameter $\rho \in [0, 1]$ that gives a pure Q-learning algorithm when $\rho = 0$ — satisfying the Bellman equation — and a pure policy search algorithm when $\rho = 1$ — locally maximising the long-term reward using the Williams' REINFORCE algorithm [Williams, 1992] described in Section 7.3. Intermediate values of ρ invoke an hybrid algorithm that is guaranteed to converge even for POMDPs.

However, it is not clear how to choose ρ given a POMDP, and because VAPS is not a pure gradient ascent it may not converge to a *locally optimal* policy when $\rho < 1$ [Sutton et al., 2000]. An alternative introduced by Sutton et al. [2000], which does converge to a locally optimal policy, incorporates a learnt Q-function directly into the gradient estimation of $\nabla \bar{J}_\beta$ or $\nabla \eta$. In this case the Q-function works like a critic aiding the actor to learn a policy [Konda and Tsitsiklis, 2000].

Both Williams' REINFORCE and VAPS were developed in the context of solving MDPs using function approximation. They can be extended to POMDPs either by assuming that observations map directly to world states — implying the assumption that a reactive policy is sufficient — and learning the policy $\mu(u|\theta, y)$, or by using the model and $\omega(b_{t+1}|\phi, b_t, y_t)$ to track belief states that are fed to the hybrid policy represented by $\mu(u_t|\theta, b_{t+1})$.

Coarse grained value estimates can be used as a seed for fine grained policy-search methods, avoiding the disadvantages of the curse of dimensionality for DP and the high variance of policy-search in the early stages. This approach is demonstrated on a real robot using a continuous time model by Roy and Thrun [2001].

Konda and Tsitsiklis [2000] present hybrid algorithms as actor-critic algorithms. The actor uses a policy-gradient approach to learn a policy as a function of a low-dimension projection of the exact value function. The projection of the value function is learnt by the critic. Convergence is shown for arbitrary state/action spaces with linear critics. The paper shows that the value function projection that should be used by the critic is determined by the how the actor is parameterised.

7 Learning Without a Model

From this point on we do not assume knowledge of the underlying MDP dynamics $q(j|i,u)$, the observation probabilities $\nu(y|i)$, or the rewards $r(i)$. To learn good policies the agent interacts with the world, comparing observation/action trajectories under different policies. With enough samples, rewards can be correlated with actions and the probability of choosing good actions increased. Section 5 briefly described Q-learning which is one example of a model-free algorithm for MDPs. It is also a starting point for model-free POMDPs.

7.1 Ignoring Hidden State

We can modify MDP Q-learning to learn the value of observations instead of states, that is, $Q(y,u)$ instead of $Q(i,u)$. This is equivalent to learning values over distributions of states instead of single states [Singh et al., 1994]. Such reactive policies generally need to be stochastic. If the agent is uncertain about the state then it may be optimal to choose actions stochastically. Examples can be constructed for which a stochastic reactive policy is arbitrarily better than the optimal deterministic reactive policy [Williams and Singh, 1999]. This method is used with stochastic-policy iteration by Jaakkola et al. [1995], converging to a local maximum when $Q(y,u)$ is parameterised by a table. A similar result is shown [Singh et al., 1994] that also has convergence results for a modified TD(0) MDP algorithm [Sutton and Barto, 1998].

In this memory-less setting $|\mathcal{G}| = 1$ making $\omega(h|\phi,g,y)$ trivial. Consequently, the policy reduces to $\mu(u|\theta,y)$.

7.2 Incorporating Memory

When the POMDP model is not available agents cannot track the belief state. The alternative to reactive policies is to use memory of the past observations and actions to resolve the hidden world state. This section describes several mechanisms for adding memory to agents.

7.2.1 HMM Based Methods

Solving POMDPs can be re-cast as the problem of determining the hidden state of the world and then applying MDP methods. A natural approach is to use

hidden Markov models which have been successfully applied to many other hidden state estimation problems [Poritz, 1988]. This approach was taken by Chrisman [1992] where HMMs are used to predict observations based on the observation/action history $\bar{y}$. A nice feature of this algorithm is the ability to grow the number of states in the HMM until performance stops improving. HMMs for predicting observations were also studied by McCallum [1996].

A problem with this approach is that the hidden state revealed by modelling *observations* is not guaranteed to reveal the state necessary to maximise the reward. One alternative studied in Aberdeen [2003] uses HMMs to model *rewards*. Also, gradient ascent of HMMs can be used to model *action* generation [Shelton, 2001a,b]. All of the HMM methods discussed in this section use a generalisation of HMMs introduced by Bengio and Frasconi [1995] called *Input/Output* HMMs (IOHMMs). The state transitions of IOHMMs can be driven by a signal other than the one being modeled.

In these models $\mathcal{B}$ is the set of reachable belief states over HMM states. This *internal-state belief* is equivalent to the forward probability over states used during Baum-Welch training of HMMs [Rabiner and Juang, 1986]. The probability of occupying each HMM state is updated using $\omega(h|\phi, g, y)$ given current assumed internal belief g and input observation y (or the previous action if observations are being modeled).

The policy $\mu(u|\theta, h, y)$ maps the belief state over HMM states to actions. For example, if the HMM is trained to emit actions then $\mu(u|\theta, h, y)$ represents the emission probabilities of HMM-state h [Shelton, 2001a]. In this case ϕ parameterises HMM transition probabilities and θ parameterises emission densities for predicted observations or generated actions. The internal-state belief update is deterministic, but the policy is stochastic. Alternatively, $\mu(u|\theta, h, y)$ can be learnt by DP, estimating values of internal-state beliefs. This is the approach used by Chrisman [1992] where the Q_{MDP} heuristic is used to assign values to internal-state beliefs.

7.2.2 Finite History Methods

Finite history methods produce policies that are functions of a finite window of past observations/actions. They have existed in the context of POMDPs at least since Brown [1972]. Window-Q uses an artificial neural network (ANN) to learn Q-values where the ANN inputs are the last n observations and actions [Lin and Mitchell, 1992]. Time-delay neural networks incorporate n steps of history into each node of the neural network [Lin, 1994]. In either case $\omega(h|\phi, g, y)$ deterministically records a finite window of $\bar{y}$, and $\mu(u|\theta, \bar{y})$ uses an ANN parameterised by θ to assign probability 1 to the action that maximises the value.

Adaptive history methods grow or shrink the number of past events that are needed to reveal the hidden state. Probabilistic Suffix Automata [Ron et al., 1994] model partially observable Markov decision processes using a tree. The history defines a path through the tree starting at the root. The root corresponds to the earliest history element. The leaf can be used to predict the next symbol, or it can be labelled with the optimal next action given the history. McCallum [1996] uses the latter approach in the UTREE algorithm (see Figure 6) to control large POMDPs such as the New York Driving problem. A statistical test is applied to leaf nodes to determine if the long-term reward can be improved by growing the tree, automatically determining the amount of memory needed.

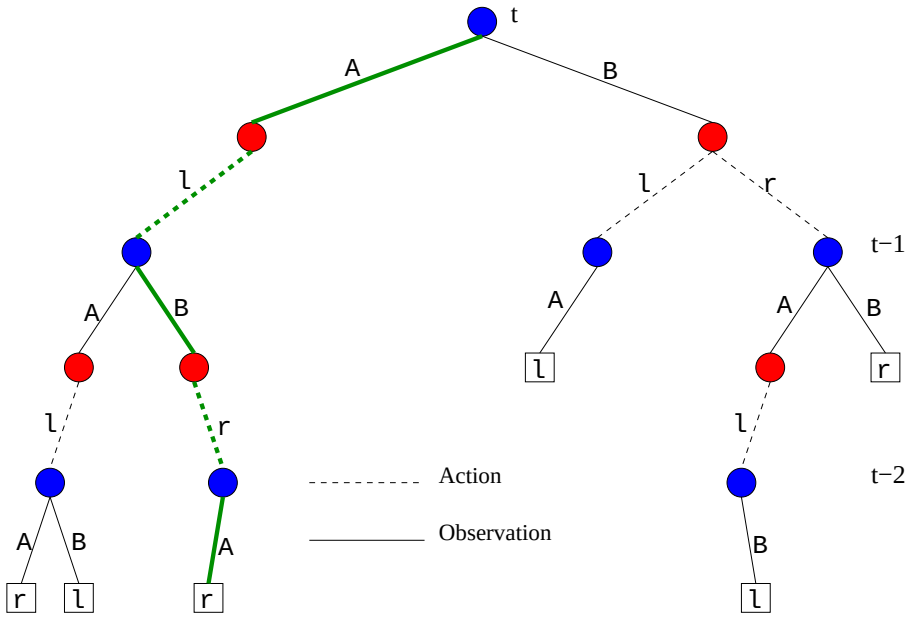


Figure 6: A history tree learnt by UTREE for some POMDP with $\mathcal{Y} = \{\texttt{A, B}\}$ and $\mathcal{U} = \{\texttt{l, r}\}$. Leaves are labelled with optimal actions and a sample path is shown for the history $\{\texttt{(A,r)(B,l)(A,}\cdot\texttt{)}\}$ choosing action `r`. The most recent observation gives the first branch to take from the root of the tree.

Observation branches can be selected by a subset of the observation vector, filtering out useless distinctions between observations. Tree methods have not been extended to continuous observation/action spaces within our knowledge. Here $\mu(u|\theta, \bar{y})$ applies the UTREE algorithm, where the tree is represented by θ. The length of history recorded is given by the maximum depth of the tree.

A similar algorithm to UTREE was described by Dutech [2000], with an enhancement for cases when a deterministic POMDP model is known. The enhancement grows the UTREE when an examination of the POMDP model determines that a path down the tree still results in an ambiguous world-state, and therefore a possibly ambiguous action.

Deterministically assigning actions to history tree leaves can lead to large trees and over-fitting in noisy worlds. Suematsu and Hayashi [1999] uses Bayesian inference to construct future reward and observation models as a function of a learnt history tree, providing a smoother model than UTREE. The policy can be constructed by performing DP on the history tree model, similar to performing DP on HMMs.

7.2.3 RNN Based Methods

Recurrent neural networks (RNNs) augment the output of an ANN with continuous state outputs that are fed back into a previous layer of the network (see Figure 7). The network has a continuous internal-state space allowing a POMDP to be controlled by presenting observations as inputs and interpreting

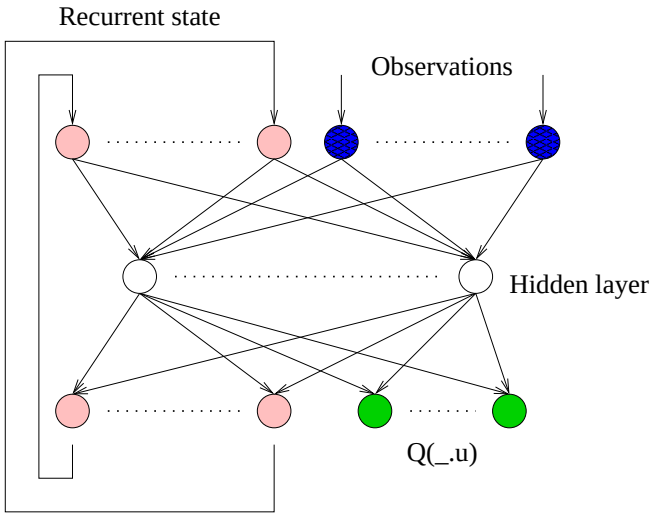


Figure 7: A simple recurrent neural network for learning Q-values.

outputs as action distributions (Recurrent-model [Lin and Mitchell, 1992]), Q-values (Recurrent-Q [Lin and Mitchell, 1992]), or by learning a second function mapping the internal state and future input predictions to actions [Schmidhuber, 1991].

The agents in this section learn both ϕ: the recurrent state model, and θ: the Q-value or action outputs. Here $\mathcal{G}$ is the set of vectors in $\mathbb{R}^n$ that are the n RNN outputs fed back to the previous layer. The internal-state update is deterministic and the actions may be chosen deterministically or stochastically.

One RNN training algorithm is back-propagation through time (BPTT) [Rumelhart et al., 1986, §8] that unfolds the network over T steps then uses the standard error back propagation algorithm for ANNs. This means that the network can only be explicitly trained to have a T step memory. However, during operation the internal state may carry relevant information for more than T steps. Real-time recurrent learning (RTRL) [Williams and Zipser, 1989] overcomes this problem by efficiently propagating errors back to time 0. These algorithms have been applied to speech recognition [Bourlard and Morgan, 1994].

Unfortunately BPTT and RTRL have difficulty learning long-term memory because the back propagated error signals tend to blow up or vanish depending on the feedback weights. Learning long term memory appears to be a problem for all model-free POMDP algorithms. History compression [Schmidhuber, 1992] uses an RNN to filter out short term information by attempting to predict its own next input as well as learning actions. When the RNN prediction fails the input is passed to a higher level RNN that concentrates on modelling long-term features of the input.

Alternatively, long short-term memory (LSTM) introduces complex neurons that learn to turn on or off their memory inputs and outputs. Another feature of LSTM is that the back-propagated error signal does not explode or vanish [Hochreiter and Schmidhuber, 1997]. LSTM has been applied to navigation

POMDPs constructed to require up to 70 steps of memory [Bakker, 2001]. The LSTM network learns Q-values and uses the interesting idea of adapting the Q-learning exploration probability by training a non-recurrent ANN to predict the variance of the Q-values.

7.2.4 Evolutionary Methods

Evolutionary methods create populations of agents and use combination and mutation rules to generate new populations. The long-term average reward is the natural fitness measure of an agent. Methods include rule based agents [Kwee et al., 2001], lookup-table agents [Moriarty et al., 1999], and probabilistic program agents [Sałustowicz and Schmidhuber, 1997]. The last two both have the advantage of allowing stochastic agents. There is a strong connection between reactive Q-learning and some evolutionary classifier algorithms [Lanzi, 1997].

Evolutionary algorithms can resolve partial observability by adding memory registers that the agent can learn to set. One successful example is a POMDP algorithm that evolves stochastic RNNs to control large POMDPs such as the New York driving scenario [Glickman and Sycara, 2001]. The hidden units of the RNN each choose an output of 1 or 0 from a distribution controlled by the squashed sum of the weighted inputs. A single output unit causes one of 3 actions to be chosen depending on which of 3 ranges the output value falls in. In this case ϕ represents the weights of the hidden units, and θ represents the weights of the output unit.

7.2.5 Finite State Controllers

The idea behind finite state controllers (FSCs) is that past events which are relevant to choosing optimal actions can be remembered indefinitely by a directed cyclic graph of internal states. Each node of the FSC is an I-state from $\mathcal{G}$. This model uses ϕ to parameterise probabilities of I-state transitions based on the current I-state and observation. The next state is chosen stochastically from the distribution $\omega(\cdot|\phi, g, y)$. Similarly, θ parameterises learnt action probabilities for each I-state, so that u is chosen stochastically from $\mu(\cdot|\theta, h, y)$.

The agent learns to use the I-states to remember only what is needed in order to act optimally. This process can be viewed as an automatic quantisation of the belief state space to provide the optimal policy *representable by* $|\mathcal{G}|$ *I-states*. As $|\mathcal{G}| \to \infty$ we can represent the optimal policy arbitrarily accurately [Bonet, 2002].

Another way to view this process is as direct learning of a stochastic *policy graph* [Meuleau et al., 1999b]. Recall from Section 6.1.3 that the nodes of a policy graph are synonymous with the I-states of an FSC. However, exact algorithms compute policy graphs with deterministic I-state transitions equivalent to a deterministic $\omega(h|\phi, g, y)$ function. Furthermore, they permit only *one action per node*. We allow stochastic I-state transitions and the policy $\mu(u|\theta, h, y)$ allows a different action distribution for each I-state *and each observation*. This means we can compute optimal policies with far fewer I-states than the equivalent policy graph.

A third way to view the process is as a grid method (recall Section 6.2.2) where the number of points is fixed, but the location of the points can move

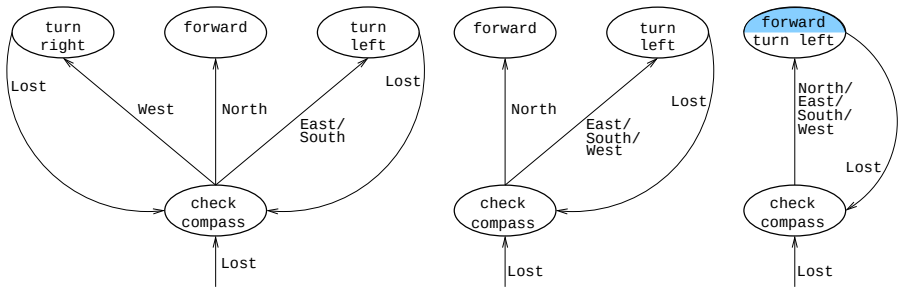


Figure 8: Figure 8(a) shows the optimal policy graph fragment for a lost agent that must move north. If the number of I-states is reduced to 3 (8(b)) then the policy is still deterministic but degraded because it may turn left 3 times instead of right once. In Figure 8(c) the agent cannot determine if it is facing north when it is in the upper I-state, thus must sometimes move forward and sometimes turn.

around to best cover the interesting regions of belief state space.

Though it is a small comfort, training an agent from the class of algorithms based on restricted FSCs has NP-hard complexity instead of the PSPACE-hard complexity of exact algorithms [Meuleau et al., 1999a].

The FSC framework encompasses other algorithms that differ by how they learn $\omega(h|\phi, g, y)$. For example, Peshkin et al. [1999] and Peshkin [2000], where the action space of SARSA or VAPS is augmented with external-memory-setting actions. Also, the algorithms of Sections 7.2.1–7.2.4 can be cast as FSC methods. FSCs can also be learnt by depth first search in constrained spaces of FSC [Meuleau et al., 1999b].

Section 7.1 discusses the necessity of stochastic policies for POMDPs in the absence of memory. Stochasticity is also needed when $|\mathcal{G}|$ is too small. When there is insufficient memory to resolve the uncertainty about the current state there is insufficient information for determining which action is best. Consider Figure 8, showing a fragment of an FSC in which a lost agent uses a compass to face north before moving on. As the number of I-states is reduced the policy degrades until for $|\mathcal{G}| = 2$ the agent cannot decide between moving forward or turning, which is a stochastic policy. Only finitely transient POMDPs (see Section 6.1.3) have optimal deterministic FSCs, that is, an optimal policy that can be represented by a policy graph.

Finite state controllers have been proposed as a useful method for fully observable MDPs with very large state spaces where the optimal reactive policy is too hard to represent. Kim et al. [2000] uses VAPS to perform policy-search in the space of FSCs where the state is fully observable but factored as described in Section 6.3.

Finite state controllers exhibit difficulties when learning non-trivial I-state functions such as those requiring long-term memory. When all I-state trajectories are nearly equally likely to occur for any world-state trajectory, there can

be little correlation between a particular I-state trajectory and a high reward, hence no single I-state trajectory is reinforced. This explanation for difficulties with long term memory was proposed for value function methods by Lanzi [2000] and termed *aliasing on the payoffs*. This problem was further analysed by Aberdeen [2003] where sufficient conditions for failure of FSC policy-gradient methods are established.

7.3 Policy-Gradient Methods

This section introduces policy-gradient methods for model-free agent training. Policy gradient methods compute (or estimate) the gradient of $\eta(\phi, \theta)$ (1) with respect to the parameters of the agent ϕ and θ. This allows η to be maximised by some form of gradient ascent procedure. In Section 6.7 we discussed some advantages of using policy-gradient methods instead of value-function methods. Another advantage is that gradient ascent is a local optimisation method. This means means we can avoid the computation complexity of exact POMDP methods. However, the trade-off is that we cannot guarantee convergence to the global maximum of $\eta(\phi, \theta)$. Consideration must also be given to the conditions under which the gradient is well defined, which generally requires the world-state Markov process to be ergodic, and the processes $\omega(h|\phi, g, y)$ and $\mu(u|\theta, h, y)$ to be differentiable.

It is easy to add domain knowledge to agents in the policy-gradient setting. Hard-wired rules can modify the distributions $\omega(h|\phi, g, y)$ and $\mu(u|\theta, h, y)$ additively, hence having no impact on the gradient computation. Domain knowledge can also be used to shape the rewards [Laud and DeJong, 2002], or to factor the observations.

Early algorithms that estimated gradients of performance measures using Monte-Carlo like methods include the *likelihood-ratio* method [Aleksandrov et al., 1968, Rubinstein, 1969]. Extensions of the likelihood-ratio method to *regenerative processes* (including Markov Decision Processes) were given by Glynn [1986, 1990], Glynn and L'Ecuyer [1995], Reiman and Weiss [1986], and Reiman and Weiss [1989]; and independently for finite-horizon MDPs by Williams [1992]. Glynn showed that the gradient could be estimated by sampling a trajectory through the state space for a T step episode of a finite-horizon MDP. The estimated gradient extended to the POMDP setting is given by

$$\widehat{\nabla \eta} = \left(\sum_{t=1}^{T} r_t \right) \sum_{t=1}^{T} \frac{\nabla \mu(u_t|\theta, y_t)}{\mu(u_t|\theta, y_t)}, \tag{10}$$

where the gradient is with respect to the policy parameters θ. This is a memoryless agent which assumes that y_t reveals sufficient world-state to act well. This scheme requires $\frac{\nabla \mu(u_t|\theta, y_t)}{\mu(u_t|\theta, y_t)}$ be well behaved. This is the case if $\mu(u_t|\theta, y_t)$ is parameterised so that $\nabla \mu(u_t|\theta, y_t)$ and $\mu(u_t|\theta, y_t)$ go to zero at the same rate. One such choice of $\mu(u_t|\theta, y_t)$ is the soft-max function. Once the gradient has been estimated a stochastic gradient ascent algorithm can be used to maximise η. Equation (10) does not take into account the causality of rewards, that is, rewards gathered *before* time t can contribute to the gradient induced by actions *after* time t. A simple fix to this is implemented by Meuleau et al. [2001], replacing the sum of rewards with the sum of rewards up to time t.

Williams' REINFORCE is unbiased. It converges to the true gradient over many episodes. Williams' REINFORCE only works for episodic tasks because the algorithm awards *equal credit* for the reward up to time t to each action. If we attempted to use Williams' REINFORCE on an infinite-horizon POMDP, using a single infinitely-long sample trajectory through state space, we cannot assign credit to specific actions because there are infinitely many of them. In order to apply Williams' REINFORCE to POMDPs it is necessary to be able to identify when the system enters a *recurrent state*, indicating the end of an episode and bounding the implications of actions on the long-term reward. This is non-trivial when the true state is masked by the observation process $\nu(y|i)$.

If the agent cannot observe visits to recurrent states, or if visits occur infrequently, we resort to biased estimates of the gradient. A number of researchers introduced discounted eligibility traces [Marbach and Tsitsiklis, 2000, Kimura et al., 1997] to bound the implications of actions, but the discount factor introduces a bias. The eligibility trace acts as memory, telling the agent what proportion of the current reward each previous action should be credited with.

The similar approach of Baxter and Bartlett [2001] is to derive, from scratch, an algorithm that directly approximates $\nabla\eta$ for infinite-horizon POMDPs. This derivation also introduces a discounted eligibility trace with discount factor $\beta \in [0, 1)$, showing that in the limit as $\beta \to 1$ the approximated gradient converges to $\nabla\eta$. The algorithm is summarised by the following expression that can be unrolled to form the core of the GPOMDP algorithm [Baxter and Bartlett, 2000, Barlett and Baxter, 2000, Baxter and Bartlett, 2001]

$$\widehat{\nabla\eta} = \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \frac{\nabla\mu(u_t|\theta, y_t)}{\mu(u_t|\theta, y_t)} \sum_{s=t+1}^{T} \beta^{s-t-1} r_s.$$

The left hand side is the approximation of the true gradient. As $\beta \to 1$ the approximation becomes perfect, but the variance goes to ∞. When $\beta < 1$ we make the assumption that the effect of actions on the reward decays exponentially with time, allowing the temporal credit assignment problem to be solved. Other methods of bounding the period in which rewards can be linked to actions are valid but the exponential decay model is easy to analyse and implement.[4] Baxter et al. [2001] shows that the variance of the estimate scales with $1/[T(1-\beta)]$, which reflects the fact that as the credit for rewards is spread over a longer time, the temporal credit assignment problem becomes harder, until the case of $\beta = 1$ gives us no discounting and the variance is infinite. The GPOMDP algorithm was proved to extend to the setting of infinite state and action spaces. An example of a continuous action space is given by Kimura and Kobayashi [1998]. Convergence *rates* for GPOMDP in terms of the POMDP *mixing time* have also been established [Bartlett et al., 2000, Bartlett and Baxter, 2002].

Williams' REINFORCE has been used as a policy-gradient algorithm for learning FSCs [Peshkin, 2000] but in its basic form is restricted to POMDPs for which a recurrent state can be identified. Aberdeen [2003] extends FSC learning to the infinite-horizon case, based on the GPOMDP algorithm.

[4] This is true of a broad class of reinforcement learning algorithms that use discount factors.

8 Further Issues

This section presents useful ideas for improving most POMDPs methods.

8.1 Variance Reduction for POMDP Methods

Monte-Carlo algorithms for POMDPs have high-variance, policy-gradient methods in particular. This section describes established methods for reducing the variance.

8.1.1 Importance Sampling

Importance Sampling (IS) is a method for improving estimates of functions of arbitrary distributions that are known only up to a normalising constant. The idea is to sample from an appropriately selected *known* distribution. Samples are weighted by the ratio of the probability under their true distribution to their probability under the sampling distribution. The sampling distribution should be our best guess of the true distribution. This means the weights reflect the relative importance of each sample of the unknown distribution. Variance reduction occurs when more weight is given to the areas of the sampling distribution where the bulk of the unknown distribution's probability mass is concentrated.

The application of IS to Monte-Carlo methods is discussed by Glynn [1996] and applied to Williams' REINFORCE and VAPS by Meuleau et al. [2000]. IS can be used to implement *off-policy* policy-gradient methods [Peshkin and Shelton, 2002], reducing the number of world interactions required as described in Section 5. These methods permit computation of bounds on the probability of finding an ϵ-approximation of the true value using a finite number of samples from the world [Peshkin and Shelton, 2002, Peshkin, 2002]. IS and Williams' REINFORCE have been used in conjunction with learning finite state controllers (Section 7.2.5) as well as reactive policies [Shelton, 2001c,b].

IS has also been extended to monitoring belief states with transitions modeled by BNs, directly minimising the variance of the belief state estimates, combining the ideas of Sections 6.3.1 and 6.6 [Ortiz and Kaelbling, 2000].

8.1.2 Reward Baselines

Marbach and Tsitsiklis [1999] use $r_t - \eta$ as a performance indicator at each time step instead of just the long-term reward η. The difference between r_t and the reward *baseline* η indicates whether the reward was above or below average. Intuitively, this indicates whether positive or negative reinforcement is required, providing variance reducing information. Weaver and Tao [2001] proved that η is the optimal baseline for policy-gradient algorithms. Greensmith et al. [2002] show that the widely used J_β baseline is *sub-optimal*. That paper also analyses the use of general additive control variates, such as actor-critic methods, to reduce variance. These ideas are similar to the actor-critic style estimates proposed by Sutton et al. [2000] and Konda and Tsitsiklis [2000].

8.1.3 Fixed Random Number Generators

A source of variance in value-function or policy-gradient estimates arises from uncertainty in the transitions, that is, it is possible to execute T policy steps

starting from the same start state and obtain different average rewards. Ng and Jordan [2000] extend the work of Kearns et al. [1999], developing the PEGASUS algorithm that is applicable to POMDPs simulated with a controlled source of randomness. It transforms arbitrary POMDPs into POMDPs with deterministic transitions by augmenting the state space with a record of the random numbers that will be used to generate the state and observation trajectories. Long-term rewards are then estimated starting from m initial states drawn at random. The m needed to obtain ϵ-convergence of the value-function scales polynomially with the discount factor and the complexity of the policy space, but not with the number of states in the POMDP. In practice this method can be applied by re-seeding the simulator's random number generator to a fixed value before each episode. PEGASUS is similar to *paired* statistical tests for policy evaluation and is compared to them under various optimisation methods by Strens and Moore [2001].

We apply the method to our Monte-Carlo policy-gradient algorithms, reducing the variance in gradient estimates. The variance reduction is not free since the method can introduce false local maxima [Aberdeen, 2003].

8.2 Multi-Agent Problems

Most algorithms discussed in this paper can be extended to multiple agents. The idea is to alter the set of actions $\mathcal{U}$ such that it contains the cross product of all the actions available to each agent, that is, for n agents, $\mathcal{U} = \{\mathcal{U}_1 \times \mathcal{U}_2 \times \cdots \times \mathcal{U}_n\}$. If the agent parameters are independent, then each agent independently chooses actions that are combined to form the meta-action. For stochastic policies, the overall action distribution is just the joint distribution of actions for each agent, $\mu(u_1, u_2, \ldots | \theta_1, \theta_2, \ldots, h_1, h_2, \ldots, y_1, y_2, \ldots)$.

Examples of multi-agent learning in the policy-gradient setting include training independent neurons in the brain [Barlett and Baxter, 1999] and training multiple network routers for maximum throughput [Tao et al., 2001]. Dutech et al. [2001] uses an on-line version of the GPOMDP algorithm to incrementally train more and more agents, in larger and larger worlds. By using value functions, multiple automated guided vehicles have been trained to operate in a factory [Makar et al., 2001].

8.3 Hierarchical POMDPs

The MAXQ algorithm extends Q-learning into tasks which have been broken down into a hierarchy of sub-tasks [Dietterich, 2000]. To achieve the full benefit of the hierarchy we also require methods to abstract states at each level so that Q-learning only proceeds on the variables relevant to that task [Dietterich, 2000]. MAXQ assumes the world is observable. In the POMDP case we can optimistically train a hierarchy of reactive policies [Wiering and Schmidhuber, 1997] or use a hierarchy of HMMs performing MAXQ on the induced state space [Theocharous et al., 2000]. Since HMMs return distributions over states a heuristic such as MLS (Section 6.2.1) is used. An alternative to HMMs is to use a hierarchy of the finite history models (Section 7.2.2), known as hierarchical suffix memory [Hernandez-Gardio and Mahadevan, 2001].

The more high-level the state model is, the more long-term the memory is at that level; which is similar to the idea of history compression (Section 7.2.3).

Useful hierarchies can be automatically determined by forming maps of POMDP states and aggregating "close" states into high level abstract states [Engel and Mannor, 2001b,a].

An hierarchy can also be useful to reason about actions at different time scales [Precup, 2000]. Ghavamzadeh and Mahadevan [2001] and Makar et al. [2001] extend MAXQ to continuous time and demonstrate the algorithm in a multi-agent automated guided vehicle setting.

9 Summary

Table 1 summarises the model-based algorithms described in this paper. Table 2 correspondingly summarises the model-free algorithms. Particular attention is paid to comparing the how the $\omega(h|\phi, g, y)$ and $\mu(u|\theta, h, y)$ processes are parameterised for each algorithm.

10 Conclusion

POMDPs offer the most general learning framework of which the authors are aware. Any causal day to day problem can be modelled as a POMDP. Finding useful algorithms, in the sense of learning reasonable policies in a timely fashion, is a tremendous and interesting challenge.

This survey has covered a small fraction of the papers in the field however the attempt was made to touch on all existing approaches to solving POMDPs. Murphy [2000] is another useful POMDP survey.

References

Douglas A. Aberdeen. *Policy-Gradient Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, Australian National Unversity, March 2003.

V. M. Aleksandrov, V. I. Sysoyev, and V. V. Shemeneva. Stochastic optimaization. *Engineering Cybernetics*, 5:11–16, 1968.

K. J. Åström. Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications*, 10, 1965.

R. Iris Bahar, Erica A. Frohm, Charles M. Gaona, and Gary D. Hachtel. Algebraic decision diagrams and their applications. In *International Conference of Computer-Aided Design*, pages 188–191. IEEE, 1993.

Leemon C. Baird and Andrew W. Moore. Gradient descent for general reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 11. MIT Press, 1999.

Bram Bakker. Reinforcement learning with LSTM in non-Markovian tasks with long-term dependencies. Technical report, Leiden University, 2001. `http://www.fsw.leidenuniv.nl/www/w3_func/bbakker/abstracts.htm`.

Table 1: A summary of the model-based algorithms described in this paper. The ω column contains `d` if the internal state update is deterministic and `s` if the update is stochastic. Similarly, the μ column indicates if the choice of action is deterministic of stochastic. Uppercase `D` indicates that the function is fixed instead of learnt. The last column describes how ϕ parameterises $\omega(h|\phi, g, y)$ and θ parameterises $\mu(u|\theta, h, y)$.

Method	ω	μ	ϕ *parameters* θ *parameters*
MDP [§5] e.g. Value Iteration	d	d	Fully observable, therefore no memory required θ stores long-term value of each state
Exact [§6.1] e.g. Incremental Pruning	D	d	I-state is b_t, ϕ represented by $q(j\|i, u)$ and $\nu(y\|i)$ Piecewise linear hyperplanes or policy graph
Heuristic [§6.2.1]	D	d	As for exact Approximation to piecewise linear hyperplanes
Grid [§6.2.2]	D	d	As for exact $\mu(u\|\theta, h, y)$ interpolates grid point values stored by θ
Factored belief [§6.3.1]	d	d	ϕ encodes *Bayesian network* or *algebraic decision tree* Could be any of the previous parameterisations
Factored value [§6.3.2]	d	d	Any of previous, including exact and factored Linear combinations of basis functions or ADDs
Planning [§6.4]	d	D	Any belief state tracking method $\mu(u\|b, y)$ searches space of future trajectories
RTDP-Bel [§6.5]	d	d	Any method to track sampled beliefs b_t θ stores value of all visited belief states
SPOVA & Linear-Q [§6.5]	d	d	Any method to track sampled beliefs b_t Smooth approx. of exact $\bar{J}_\beta$ learnt by gradient ascent
Particle Filters [§6.6]	d	d	ϕ represents tracked b_t as n particles KNN to infer value for b_t represented by particles
Policy Iteration [§6.7]	d	d	ϕ is a policy graph (PG) converted to $\bar{J}_\beta$ for learning steps θ maps policy graph nodes to actions, learnt during DP
Depth first PG search [§6.7]	d	d	ϕ chosen by search of a tree of constrained PGs θ maps PG nodes to actions
Gradient ascent of PGs [§6.7]	s	s	ϕ is PG transition probs. learnt by gradient ascent θ stochastically maps PG nodes to actions
Approx. Gradient PG [§6.7]	s	s	PG transition probabilities controlled by ANN ANN maps PG nodes to action probabilities
Actor-Critic & VAPS[§ 6.8]	D	s	Usually no internal state θ is policy learnt by gradient ascent *and* value-iteration

Table 2: A summary of the model-free algorithms described in this paper. Each column has the same meaning as Table 1. The tables are not a complete summary of all POMDP algorithms.

Method	ω	μ	ϕ *parameters* θ *parameters*
JSJ Q-learning [§7.1]	D	s	Assume $y_t = i_t$, so no internal state θ stores long-term values of each observation
HMM methods [§7.2.1]	d	s	ϕ is HMM transition probabilities θ is action probs. or value of HMM belief states
Window-Q [§7.2.2]	D	d	ϕ deterministically records last n observations $\bar{y}$ θ is ANN weights mapping $\bar{y}$ to values
UTREE [§7.2.2]	D	d	ϕ deterministically records last n observations $\bar{y}$ θ represents tree; follow $\bar{y}$ branch to get u_t
RNNs [§7.2.3]	d	d	RNN maps y_t & RNN state output to new state output RNN maps y_t & RNN state output to actions or values
Evolutionary [§7.2.4]	s	D	ϕ is RNN trained using EAs & stochastic sigmoid outputs θ weights sigmoid outputs to select actions
FSCs [§7.2.5]	s	s	ϕ is prob. of I-state transition $g \rightarrow h$ θ is prob. of u_t given I-state h
Williams' REINFORCE [§7.3]	s	s	I-states may be changed by memory setting actions Grad ascent of θ that maps $y_t \rightarrow u_t$
GPOMDP [§7.3]	s	s	I-state FSC learnt via gradient Grad ascent of θ that maps $y_t \rightarrow u_t$ for infinite-horizons

Peter L. Barlett and Jonathan Baxter. Hebbian synaptic modifications in spiking neurons that learn. Technical report, Computer Sciences Laboratory, RSISE, ANU, November 1999.

Peter L. Barlett and Jonathan Baxter. Estimation and approximation bounds for gradient based reinforcment learning. In *Thirteenth Annual Conference on Computational Learning Theory*, 2000. `http://discus.anu.edu.au/~bartlett/`.

Peter Bartlett and Jonathan Baxter. Estimation and approximation bounds for gradient-based reinforcement learning. *Journal of Computer and System Sciences*, 2002. To appear.

Peter L. Bartlett, Stephane Boucheron, and Gabor Lugosi. Model selection and error estimation. Technical report, Computer Sciences Laboratory, RSISE, ANU, 2000.

Jonathan Baxter and Peter L. Bartlett. Reinforcement learning in POMDP's via direct gradient ascent. In *Proc. 17th International Conf. on Machine Learning*, pages 41–48. Morgan Kaufmann, San Francisco, CA, 2000.

Jonathan Baxter and Peter L. Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350, 2001.

Jonathan Baxter, Peter L. Bartlett, and Lex Weaver. Experiments with infinite-horizon, policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:351–381, 2001.

R. Bellman, R. Kalaba, and B. Kotkin. Polynomial approximation — a new computational technique in dynamic programming: allocation processes. *Mathematics of Computation*, 17:155–161, 1963.

Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, N.J., 1957.

Yoshua Bengio and Paolo Frasconi. An input output HMM architecture. In *Advances in Neural Information Processing Systems*, volume 7, pages 427–434. The MIT Press, 1995. URL `citeseer.nj.nec.com/bengio95input.html`.

Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dymanic Programming*. Athena Scientific, 1996.

Jim Blythe. Decision-theoretic planning. *AI Magazine*, 1(20), Summer 1999. `http://www.isi.edu/~blythe/papers/aimag.html`.

Blai Bonet. An ϵ-optimal grid-based algorithm for partially observable Markov decision processes. In *19th International Conference on Machine Learning*, Sydney, Australia, June 2002.

Herv A. Bourlard and Nelson Morgan. *Connectionist Speech Recognition* A Hybrid Approach. Kluwer Academic Publishers, 1st edition, 1994.

C. Boutilier and D. Poole. Computing optimal policies for partially observable decision processes using compact representations, 1996.

Xavier Boyen and Daphne Koller. Tractable inference for complex stochastic processes. In *Conference on Uncertainty in Artificial Intelligence*, pages 33–42, 1998. URL `citeseer.nj.nec.com/boyen98tractable.html`.

Ronen I. Brafman. A heuristic variable grid solution method for POMDPs. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI '97)*, 1997.

George W. Brown. *Statistical Papers in Honor of George W. Snedecor*, chapter Recursive Sets of Rules in Statistical Decision Processes, pages 59–76. Iowa State University Press, Ames, Iowa, 1972. ISBN 0-8138-1585-1.

Anthony Cassandra. *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, Brown University, May 1998.

Anthony R. Cassandra. POMDPs for Dummies: POMDPs and their algorithms, sans formula, January 1999. `http://www.cs.brown.edu/research/ai/pomdp/tutorial/index.html`.

Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Articial Intelligence*, 1994.

Hsien-Te Cheng. *Algorithms for Parially Observable Markov Decision Processes*. PhD thesis, University of British Columbia, British Columbia, Canada, 1988.

Lonnie Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *National Conference on Artificial Intelligence*, pages 183–188, 1992.

T. Dean and K. Kanazawa. A model for reasoning about presistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Roayal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

Thomas G. Dietterich. An overview of MAXQ hierarchical reinforcement learning. In *SARA*, pages 26–44, 2000. URL `citeseer.nj.nec.com/dietterich00overview.html`.

D. Draper, S. Hanks, and D. Weld. A probabilistic model of action for least-commitment planning with information gathering. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 178–186. Morgan Kaufmann, 1994.

Alain Dutech. Solving POMDPs using selected past events. In *Proceedings of the 14th European Conference on Articial Intelligence, ECAI2000*, pages 281–285, 2000. URL `citeseer.nj.nec.com/dutech00solving.html`.

Alain Dutech, Olivier Buffet, and Francois Charpillet. Multi-agent systems by incremental gradient reinforcement learning. In *Proceedings of the Seventeenth International Joint Conference on Articial Intelligence, IJCAI-01*, pages 833–838, Seattle, WA, 4–10 August 2001. URL `citeseer.nj.nec.com/dutech01multiagent.html`.

Yaalov Engel and Shie Mannor. Learning embedded maps of Markov processes. In *The Eighteenth International Conference on Machine Learning*, June 2001a.

Yaalov Engel and Shie Mannor. Learning embedded maps of Markov processes. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 138–145. Morgan Kaufmann, June 2001b.

Paul Fearnhead. *Sequential Monte Carlo methods in filter theory*. PhD thesis, Merton College, University of Oxford, 1998.

Héctor Geffner and Blai Bonet. Solving large POMDPs by real time dynamic programming. Working Notes Fall AAAI Symposium on POMDPs, 1998. `http://www.cs.ucla.edu/~bonet/`.

Mohammad Ghavamzadeh and Sridhar Mahadevan. Continuous-time hierarchical reinforcement learning. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 186–193. Morgan Kaufmann, June 2001.

Matthew R. Glickman and Katia Sycara. Evolutionary search, stochastic policies with memory, and reinforcement learning with hidden state. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 194–201. Morgan Kaufmann, June 2001.

Peter W Glynn. Stochastic approximation for Monte-Carlo optimization. In *Proceedings of the 1986 Winter Simulation Conference*, pages 356–365, 1986.

Peter W Glynn. Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33:75–84, 1990.

Peter W. Glynn. Importance sampling for Monte Carlo estimation of quantiles. Technical report, Dept. of Operations Research, Stanford University, 1996. URL `citeseer.nj.nec.com/glynn96importance.html`.

Peter W Glynn and Paul L'Ecuyer. Likelihood ratio gradient estimation for regenerative stochastic recursions. *Advances in Applied Probability, 27, 4 (1995)*, 27:1019–1053, 1995.

Evan Greensmith, Peter L. Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 14, Vancouver, BC, December 2002. MIT Press.

Carlon Guestrin, Daphne Killer, and Ronald Parr. Solving factored POMDPs with linear value functions. In *IJCAI-01 workshop on Palling under Uncertainty and Incomplete Information*, Seattle, Washington, August 2001. URL `citeseer.nj.nec.com/440372.html`.

Eric A. Hansen. Solving POMDPs by searching in policy space. In *The Eighth Conference on Uncertainty in Artificial Intelligence*, pages 211–219, Madison, WI, 1998. URL `citeseer.nj.nec.com/hansen98solving.html`.

Eric A. Hansen and Zhengzhu Feng. Dynamic programming for POMDPs using a factored state representation. In *The Fifth International Conference on Artificial Intelligence Planning and Scheduling*, pages 130–139, Breckenridge, Colarado, April 2000. URL `citeseer.nj.nec.com/hansen00dynamic.html`.

Milos Hauskrecht. Incremental methods for computing bounds in partially observable Markov decision processes. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97)*, pages 734–739, Providence, Rhode Island, 1997. MIT Press. ISBN 0-262-51095-2.

Milos Hauskrecht. Value-function approximations for partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, 13: 33–94, August 2000.

Natialia Hernandez-Gardio and Sridhar Mahadevan. Hierarchical memory-based reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 13, 2001.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. URL `citeseer.nj.nec.com/hochreiter95long.html`.

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

Tommi Jaakkola, Satinder P. Singh, and Michael I. Jordan. Reinforcement learning algorithm for partially observable Markov decision problems. In *Advances in Neural Information Processing Systems*, volume 7, pages 345–352. The MIT Press, 1995. URL `citeseer.nj.nec.com/jaakkola95reinforcement.html`.

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

Michael J. Kearns, Yishay Mansour, and Andrew Y. Ng. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. In *IJCAI*, pages 1324–1231, 1999. URL `citeseer.nj.nec.com/kearns99sparse.html`.

Kee-Eung Kim, Thomas Dean, and Nicolas Meuleau. Approximate solutions to factored Markov decision processes via greedy search in the space of finite state controllers. In *Artificial Intelligence Planning Systems*, pages 323–330, 2000. URL `citeseer.nj.nec.com/392137.html`.

Hajime Kimura and Shigenobu Kobayashi. Reinforcement learning for continuous action using stochastic gradient ascent. In *Intelligent Autonomous Systems (IAS-5)*, pages 288–295, 1998.

Hajime Kimura, Kazuteru Miyazaki, and Shigenobu Kobayashi. Reinforcement learning in POMDPs with function approximation. In *Proc. 14th International Conference on Machine Learning*, pages 152–160. Morgan Kaufmann, 1997.

Daphne Koller and Ronald Parr. Policy iteration for factored MDPs, 2000. URL `citeseer.nj.nec.com/koller00policy.html`.

V. Konda and J. Tsitsiklis. Actor-critic algorithms, 2000. URL citeseer.nj.nec.com/434910.html.

Ivo Kwee, Marcus Hutter, and Jürgen Schmidhuber. Market-based reinforcement learning in partially observable worlds. In *Proceedings of the 11th International Conference on Artificial Neural Networks*. Springer-Verlag, August 2001. URL citeseer.nj.nec.com/440786.html.

Pier Luca Lanzi. Solving problems in partially observable environments with classifer systems. Technical Report N. 97.45, Dipartimento di Elettronica e Informazione, Politecnico di Milano, October 1997.

Pier Luca Lanzi. Adaptive agents with reinforcement learning and internal memory. In *The Sixth International Conference on the Simulation of Adaptive Behavior (SAB2000)*, 2000. URL citeseer.nj.nec.com/346913.html.

Adam Laud and Gerald DeJong. Reinforcement learning and shaping: Encouraging intended behaviors. In *Proceedings of the 19th International Conference on Machine Learning*, Syndey, Australia, 2002. Morgan Kaufmann.

D-T. Lin. *The Adaptive Time-Delay Neural Network: Characterization and Applications to Pattern Recognition, Prediction and Signal Processing*. PhD thesis, Institute for Systems Research, University of Maryland, 1994. http://www.isr.umd.edu/TechReports/ISR/1994/PhD_94-12/PhD_94-12.phtml.

Long-Ji Lin and Tom M. Mitchell. Memory approaches to reinforcement learning in non-Markovian domains. Technical Report CS-92-138, Carnegie Mellon, Pittsburgh, PA, 1992. http://citeseer.nj.nec.com/lin92memory.html.

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, San Francisco, CA, 1995. Morgan Kaufmann. URL citeseer.nj.nec.com/littman95learning.html.

W. S. Lovejoy. A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research*, 28(1):47–65, 1991.

Christopher Lusena, Judy Goldsmith, and Martin Mundhenk. Nonapproximability results for partially observable Markov decision processes. *Journal of Artificial Intelligence*, 14:83–103, March 2001.

O. Madani, S. Hanks, and A. Condon. On the undecidability of probabilistic planning and infinite-horizon paritally observable Markov decision problems. In *Proceedings poof the 16th National Conference of Artifical Intelligence*, pages 541–548, 1999.

R. Makar, S. Mahadevan, and M. Ghavamzadeh. Hierarchical multiagent reinforcement learning. In *Proceedings of the Fifth International Conference on Autonomous Agents*, Montréal, Canada, May 2001. ACM. citeseer.nj.nec.com/article/makar01hierarchical.html.

Perter Marbach and John N. Tsitsiklis. Gradient-based optimisation of Markov reward processes: Practical variants. In *38th IEEE Conference on Decisions and Control*, December 1999.

Perter Marbach and John N. Tsitsiklis. Gradient-based optimisation of Markov reward processes: Practical variants. Technical report, Center for Communications Systems Research, University of Cambridge, March 2000.

David A. McAllester and Satinder Singh. Approximate planning for factored POMDPs using belief state simplification. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 409–416, 1999.

Andrew Kachites McCallum. *Reinformcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, 1996. `http://www.cs.rochester.edu/u/mccallum/phd-thesis/`.

Nicolas Meuleau, Kee-Eung Kim, Leslie Pack Kaelbling, and Anthony R. Cassandra. Solving POMDPs by searching the space of finite policies. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 127–136. Computer Science Dept., Brown University, Morgan Kaufmann, July 1999a.

Nicolas Meuleau, Leonid Pechkin, Leslie P. Kaelbling, and Kee-Eung Kim. Off-policy policy search. Technical report, MIT Artificial Intelligence Laboratory and Brown University, 2000.

Nicolas Meuleau, Leonid Peshkin, and Kee-Eung Kim. Exploration in gradient-based reinforcement learning. AI Memo 2001-003, MIT, April 2001. `http://www.ai.mit.edu/research/publications/2001-publications.shtml`.

Nicolas Meuleau, Leonid Peshkin, Kee-Eung Kim, and Leslie Pack Kaelbling. Learning finite-state controllers for partially observable environments. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*. Computer Science Dept., Brown University, Morgan Kaufmann, July 1999b.

Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

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

David E. Moriarty, Alan C. Schultz, and John J. Grefenstette. Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, (11):199–229, August 1999.

Kevin P. Murphy. A survey of POMDP solution techniques. Technical report, Dept. of Computer Science, U.C.Berkeley, September 2000. `http://www.cs.berkeley.edu/~murphyk/publ.html`.

A. Ng and M. Jordan. PEGASUS: A policy search method for large MDPs and POMDPs. In *UAI, 2000*, 2000. URL `citeseer.nj.nec.com/297555.html`.

N. J. Nilsson. *Principles of Artificial Intelligence*. Tioga Publishing Company, 1980.

I. Nourbakhsh, R. Powers, and S. Birchfield. DERVISH an office-navigating robot. *AI Magazine*, 16(2):53–60, 1995.

Luis E. Ortiz and Leslie Pack Kaelbling. Adaptive importance sampling for estimation in structured domains. In *Proceedings of the Sixteenth Annual Conference on Uncertainty in Articial Intelligence (UAI2000)*, pages 446–454. Morgan Kaufmann Publishers, 2000.

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

Ronald Parr and Stuart Russell. Approximating optimal policies for partially observable stochastic domains. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1088–1094. Morgan Kaufmann, 1995. URL `citeseer.nj.nec.com/parr95approximating.html`.

Mark D. Pendrith and Michael J. McGarity. An analysis of direct reinforcement learning in non-Markovian domains. In *Proc. 15th International Conf. on Machine Learning*, pages 421–429. Morgan Kaufmann, San Francisco, CA, 1998. URL `citeseer.nj.nec.com/11788.html`.

Leonid Peshkin. *Policy Search for Reinforcement Learning*. PhD thesis, Brown University, 2002. Draft.

Leonid Peshkin, Nicolas Meuleau, and Leslie Kaelbling. Learning policies with external memory. In *Proceedings of the Sixteenth International Conference in Machine Learning*, pages 307–314. Morgan Kaufmann, 1999.

Leonid Peshkin and Christian R. Shelton. Learning from scarce experience. In *Proceedings of ICML*, number 19, Sydney, Australia, 2002. `http://www.ai.mit.edu/~pesha/Public/papers.html`.

Leonid M. Peshkin. Thesis proposal: Architectures for policy search. `http://www.ai.mit.edu/~pesha/Public/papers.html`, July 2000.

Alan B. Poritz. Hidden Markov models: A guided tour. In *ICASSP '88*, pages 7–13. Morgan Kaufmann, 1988.

Pascal Poupart and Craig Boutilier. Value-directed belief state approximation for POMDPs. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 409–416, 2000. `http://www.cs.toronto.edu/~cebly/papers.html`.

Pascal Poupart and Craig Boutilier. Vector-space analysis of belief-state approximation for POMDPs. In *Uncertainty in Artificial Intelligence 2001*, August 2001.

Pascal Poupart, Luis E. Ortiz, and Craig Boutilier. Value-directed sampling methods for monitoring POMDPs. In *Uncertainty in Artificial Intelligence 2001*, August 2001. URL `citeseer.nj.nec.com/445996.html`.

Doina Precup. *Temporal Abstraction in Reinforcement Learning*. PhD thesis, University of Massachusetts, may 2000. URL `citeseer.nj.nec.com/precup00temporal.html`.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley series in probability and mathematical statistics. Wiley, 1994. ISBN 0-471-61977-9.

L. R. Rabiner and B. H. Juang. An introduction to hidden Markov models. *IEEE ASSP Magazine*, pages 4–16, January 1986.

M I Reiman and A Weiss. Sensitivity analysis via likelihood ratios. In *Proceedings of the 1986 Winter Simulation Conference*, 1986.

M I Reiman and A Weiss. Sensitivity analysis for simulations via likelihood ratios. *Operations Research*, 37, 1989.

Andrés Rodríguez, Ronald Parr, and Daphne Koller. Reinforcement learning using approximate belief states. In *Advances in Neural Information Processing Systems*, volume 12, 2000.

Dana Ron, Yoram Singer, and Naftali Tishby. The power of amnesia. In *Advances in Neural Information Processing Systems*, volume 6, pages 176–183. Morgan Kaufmann Publishers, Inc., 1994. URL `citeseer.nj.nec.com/ron94power.html`.

Nicholas Roy and Sebastian Thrun. Integrating value functions and policy search for continuous Markov decision processes. In *Advances in Neural Information Processing Systems*, volume 14, 2001.

Reuven Y. Rubinstein. *Some Problems in Monte Carlo Optimization*. PhD thesis, 1969.

D. Rumelhart, G. Hinton, and R. R. Williams. *Parallel Distributed Processing*, chapter Learning internal representations by error propagation. MIT Press, Cambridge, MA., 1986.

Brian Sallans. Learning factored representations for partially observable Markov decision processes. In *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 2000. URL `citeseer.nj.nec.com/sallans00learning.html`.

Rafał Sałustowicz and Jürgen Schmidhuber. Probabilistic incremental program evolution. *Evolutionary Computation*, 5(2):123–141, 1997.

J. H. Schmidhuber. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 2(4):234–242, 1992.

Jürgen Schmidhuber. Reinforcement learning in Markovian and non-Markovian environments. In *Advances in Neural Information Processing Systems*, volume 3, pages 500–506. Morgan Kaufmann Publishers, Inc., 1991. URL `citeseer.nj.nec.com/100953.html`.

Christian Shelton. *Importance Sampling for Reinforcement Learning with Multiple Objectives*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, August 2001a. `http://robotics.stanford.edu/~cshelton/papers/`.

Christian R. Shelton. Policy improvement for POMDPs using normalized importance sampling. In *Uncertainty in Artificial Intelligence*, August 2001b.

Christian R. Shelton. Policy improvment for POMDPs using normalized importance sampling. Technical Report AI Memo 2001-002, MIT, Cambridge, MA, March 2001c. `http://www.ai.mit.edu/people/cshelton/papers/`.

Reid Simmons and Sven Koenig. Probabilistic robot navigation in partially observable environments. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1080–1087, 1995. URL `citeseer.nj.nec.com/article/simmons95probabilistic.html`.

S. Singh, T. Jaakkola, and M. Jordan. Learning without state-estimation in partially observable Markovian decision processes. In *Proceedings of ICML 1994*, number 11, 1994.

Satinder P. Singh, Tommi Jaakkola, and Michael I. Jordan. Reinforcement learning with soft state aggregation. In *Advances in Neural Information Processing Systems*, volume 7, pages 361–368. The MIT Press, 1995. URL `citeseer.nj.nec.com/article/singh95reinforcement.html`.

R D Smallwood and Edward J Sondik. The optimal control of partially observable Markov decision processes over a finite horizon. *Operations Research*, 21 (5):1071–1088, 1973.

Edward J. Sondik. *The Optimal Control of Paritally Observable Markov Decision Processes*. PhD thesis, Stanford University, Standford, CA., 1971.

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

Malcom J. A. Strens and Andrew W. Moore. Direct policy search using paired statistical tests. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 545–552. Morgan Kaufmann, June 2001.

N. Suematsu and A. Hayashi. A reinforcement learning algorithm in partially observable environments using short-term memory. In *Advances in Neural Information Processing Systems*, volume 11, pages 1059–1065, 1999. URL `citeseer.nj.nec.com/suematsu99reinforcement.html`.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge MA, 1998. ISBN 0-262-19398-1.

Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems*, 12:1057–1063, 2000.

Richard S. Sutton, Satinder Singh, Doina Precup, and Balaraman Ravindran. Improved switching among temporally abstract actions. In *Advances in Neural Information Processing Systems*. MIT Press, 1999.

Nigel Tao, Jonathan Baxter, and Lex Weaver. A multi-agent, policy-gradient approach to network routing. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 553–560. Morgan Kaufmann, June 2001.

Georgios Theocharous, Khashayar Rohanimanesh, and Sridhar Mahadevan. Learning and planning with hierarchical stochastic models for robot navigation. In *ICML 2000 Workshop on Machine Learning of Spatial Knowledge*, Stanford, 2000.

Sebastian Thrun. Monte Carlo POMDPs. In *Advances in Neural Information Processing Systems 12*. MIT Press, 2000. `http://citeseer.nj.nec.com/thrun99monte.html`.

R. Washington. BI-POMDP: Bounded incremental partially-observable Markov-model planning. In *Proceedings of the Fourth European Conference on Planning*, Toulouse, France, September 1997. Springer-Verlag.

Lex Weaver and Nigel Tao. The optimal reward baseline for gradient-based reinforcement learning. In *Uncertainty in Artificial Intelligence: Proceedings of the Seventeenth Conference (2001)*, pages 538–545, Seattle, WA, 2–5 August 2001. Morgan Kaufman.

Marco Wiering and Jürgen Schmidhuber. HQ-Learning. *Adaptive Behaviour*, 6 (2), 1997.

John K. Williams and Satinder Singh. Experimental results on learning stochastic memoryless policies for partially observable Markov decision processes. In *Advances in Neural Information Processing Systems*, volume 11, 1999.

R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural network. *Neural Computation*, 1(2):270–280, 1989.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

Nevin L. Zhang. Efficient planning in stochastic domains through exploiting problem characteristics. Technical Report HKUST-CS95-40, Dept. of Computer Science, Hong Kong University of Science and Technology, August 1995.

Nevin L. Zhang and Wenju Liu. Planning in stochastic domains: Problem characteristics and approximation. Technical Report HKUST-CS96-31, Dept. of Computer Science, Hong Kong University of Science and Technology, 1996. URL `citeseer.nj.nec.com/article/zhang96planning.html`.

Nevin L. Zhang and Weihong Zhang. Speeding up the convergence of value iteration in partially observable Markov decision processes. *Journal of Artificial Intelligence*, 14:29–51, Febrauary 2001.

Valentina Bayer Zubeck and Thomas G. Dietterich. A POMDP approximation algorithm that anticipates the need to observe. In *Pacific Rim International Conference on Artificial Intelligence*, pages 521–532, 2000. URL `citeseer.nj.nec.com/bayer00pomdp.html`.