Acting Optimally in Partially Observable Stochastic Domains

Anthony R. Cassandra[*], **Leslie Pack Kaelbling**[†] and **Michael L. Littman**[‡]
Department of Computer Science
Brown University
Providence, RI 02912
{arc,lpk,mll}@cs.brown.edu

Abstract

In this paper, we describe the partially observable Markov decision process (POMDP) approach to finding optimal or near-optimal control strategies for partially observable stochastic environments, given a complete model of the environment. The POMDP approach was originally developed in the operations research community and provides a formal basis for planning problems that have been of interest to the AI community. We found the existing algorithms for computing optimal control strategies to be highly computationally inefficient and have developed a new algorithm that is empirically more efficient. We sketch this algorithm and present preliminary results on several small problems that illustrate important properties of the POMDP approach.

Introduction

Agents that act in real environments, whether physical or virtual, rarely have complete information about the state of the environment in which they are working. It is necessary for them to choose their actions in partial ignorance and often it is helpful for them to take explicit steps to gain information to achieve their goals most efficiently.

This problem has been addressed in the artificial intelligence (AI) community using formalisms of epistemic logic and by incorporating knowledge preconditions and effects into their planners (Moore 1985). These solutions are applicable to fairly high-level problems in which the environment is assumed to be completely deterministic, an assumption that often fails in low-level control problems.

Domains in which actions have probabilistic results and the agent has direct access to the state of the environment can be formalized as Markov decision processes (MDPs) (Howard 1960). An important aspect of the MDP model is that it provides the basis for algorithms that provably find optimal policies (mappings from environmental states to actions) given a stochastic model of the environment and a goal. MDP models play an important role in current AI research on planning (Dean *et al.* 1993; Sutton 1990) and learning (Barto, Bradtke, & Singh 1991; Watkins & Dayan 1992), but the assumption of complete observability provides a significant obstacle to their application to real-world problems.

This paper explores an extension of the MDP model to *partially observable Markov decision processes* (POMDPs) (Monahan 1982; Lovejoy 1991), which, like MDPs, were developed within the context of operations research. The POMDP model provides an elegant solution to the problem of acting in partially observable domains, treating actions that affect the environment and actions that only affect the agent's state of information uniformly. We begin by explaining the basic POMDP formalism; next we present an algorithm for finding arbitrarily good approximations to optimal policies and a method for the compact representation of many such policies; finally, we conclude with examples that illustrate generalization in the policy representation and taking action to gain information.

Partially Observable Markov Decision Processes

Markov Decision Processes An MDP is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$, where $\mathcal{S}$ is a finite set of environmental states that can be reliably identified by the agent; $\mathcal{A}$ is a finite set of actions; T is a state transition model of the environment, which is a function mapping elements of $\mathcal{S} \times \mathcal{A}$ into discrete probability distributions over $\mathcal{S}$; and R is a *reward function* mapping $\mathcal{S} \times \mathcal{A}$ to the real numbers that specify the instantaneous reward that the agent derives from taking an action in a state. We write $T(s, a, s')$ for the probability that the environment will make a transition from state s to state s' when action a is taken and we write $R(s, a)$ for the immediate reward to the agent for taking action a in state s. A *policy*, π, is a mapping from $\mathcal{S}$ to $\mathcal{A}$, specifying

[*]Anthony Cassandra's work was supported in part by National Science Foundation Award IRI-9257592.

[†]Leslie Kaelbling's work was supported in part by a National Science Foundation National Young Investigator Award IRI-9257592 and in part by ONR Contract N00014-91-4052, ARPA Order 8225.

[‡]Michael Littman's work was supported by Bellcore.

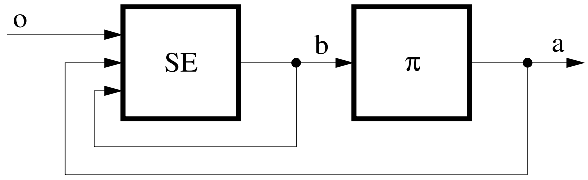

Figure 1: Controller for a POMDP

an action to be taken in each situation.

Adding Partial Observability When the state is not completely observable, we must add a model of observation. This includes a finite set, $\mathcal{O}$, of possible observations and an observation function, O, mapping $\mathcal{A} \times \mathcal{S}$ into discrete probability distributions over $\mathcal{O}$. We write $O(a, s, o)$ for the probability of making observation o from state s after having taken action a.

One might simply take the set of observations to be the set of states and treat a POMDP as if it were an MDP. The problem is that the process would not necessarily be Markov since there could be multiple states in the environment that require different actions but appear identical. As a result, even an optimal policy of this form can have arbitrarily poor performance.

Instead, we introduce a kind of internal state for the agent. A *belief state* is a discrete probability distribution over the set of environmental states, $\mathcal{S}$, representing for each state the probability that the environment is currently in that state. Let $\mathcal{B}$ be the set of belief states. We write $b(s)$ for the probability assigned to state s when the agent's belief state is b.

Now, we can decompose the problem of acting in a partially observable environment as shown in Figure 1. The component labeled "SE" is the *state estimator*. It takes as input the last belief state, the most recent action and the most recent observation, and returns an updated belief state. The second component is the *policy*, which now maps belief states into actions.

The state estimator can be constructed from T and O by straightforward application of Bayes' rule. The output of the state estimator is a belief state, which can be represented as a vector of probabilities, one for each environmental state, that sums to 1. The component corresponding to state s', written $\mathrm{SE}_{s'}(b, a, o)$, can be determined from the previous belief state, b, the previous action, a, and the current observation, o, as follows:

$$\begin{aligned}\mathrm{SE}_{s'}(b, a, o) &= \Pr(s' \mid a, o, b) \\ &= \frac{\Pr(o \mid s', a, b) \Pr(s' \mid a, b)}{\Pr(o \mid a, b)} \\ &= \frac{O(a, s', o) \sum_{s \in \mathcal{S}} T(s, a, s') b(s)}{\Pr(o \mid a, b)}\end{aligned}$$

where $Pr(o \mid a, b)$ is a normalizing factor defined as

$$\Pr(o \mid a, b) = \sum_{s' \in \mathcal{S}} O(a, s', o) \sum_{s \in \mathcal{S}} T(s, a, s') b(s) \ .$$

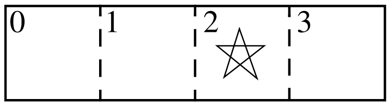

Figure 2: A Simple POMDP environment

The resulting function will ensure that our current belief accurately summarizes all available information.

Example A simple example of a POMDP is shown in Figure 2. It has four states, one of which (state 2) is designated as the goal state. An agent is in one of the states at all times; it has two actions, left and right, that move it one state in either direction. If it moves into a wall, it stays in the state it was in. If the agent reaches the goal state, no matter what action it takes, it is moved with equal probability into state 0, 1, or 3 and receives reward 1. This problem is trivial if the agent can observe what state it is in, but is more difficult when it can only observe whether or not it is currently at the goal state.

When the agent cannot observe its true state, it can represent its belief of where it is with a probability vector. For example, after leaving the goal, the agent moves to one of the other states with equal probability. This is represented by a belief state of $\langle \frac{1}{3}, \frac{1}{3}, 0, \frac{1}{3} \rangle$. After taking action "right" and not observing the goal, there are only two states from which the agent could have moved: 0 and 3. Hence, the agent's new belief vector is $\langle 0, \frac{1}{2}, 0, \frac{1}{2} \rangle$. If it moves "right" once again without seeing the goal, the agent can be sure it is now in state 3 with belief state $\langle 0, 0, 0, 1 \rangle$. Because the actions are deterministic in this example, the agent's uncertainty shrinks on each step; in general, some actions in some situations will decrease the uncertainty while others will increase it.

Constructing Optimal Policies

Constructing an optimal policy can be quite difficult. Even specifying a policy at every point in the uncountable state space is challenging. One simple method is to find the optimal state-action value function, Q^*_{CO}, for the completely observable MDP $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$ (Watkins & Dayan 1992); then, given belief state b as input, generate action $\mathrm{argmax}_{a \in \mathcal{A}} \sum_s b(s) Q^*_{\mathrm{CO}}(s, a)$. That is, act as if the uncertainty will be present for one action step, but that the environment will be completely observable thereafter. This approach, similar to one used by Chrisman (Chrisman 1992), leads to policies that do not take actions to gain information and will therefore be suboptimal in many environments.

The key to finding truly optimal policies in the partially observable case is to cast the problem as a *completely observable* continuous-space MDP. The state set of this "belief MDP" is $\mathcal{B}$ and the action set is $\mathcal{A}$. Given a current belief state b and action a, there are only

$|\mathcal{O}|$ possible successor belief states b', so the new state transition function, τ, can be defined as

$$\tau(b, a, b') = \sum_{\{o \in \mathcal{O} | \mathrm{SE}(b,a,o)=b'\}} \Pr(o \mid a, b) \ ,$$

where $\Pr(o \mid a, b)$ is defined above. If the new belief state, b', cannot be generated by the state estimator from b, a, and some observation, then the probability of that transition is 0. The reward function, ρ, is constructed from R by taking expectations according to the belief state; that is,

$$\rho(b, a) = \sum_{s \in \mathcal{S}} b(s) R(s, a) \ .$$

At first, this may seem strange; it appears the agent is rewarded simply for *believing* it is in good states. Because of the way the state estimation module is constructed, it is not possible for the agent to purposely delude itself into believing that it is in a good state when it is not.

The belief MDP is Markov (Astrom 1965), that is, having information about previous belief states cannot improve the choice of action. Most importantly, if an agent adopts the optimal policy for the belief MDP, the resulting behavior will be optimal for the partially observable process. The remaining difficulty is that belief space is continuous; the established algorithms for finding optimal policies in MDPs work only in finite state spaces. In the following sections, we discuss the method of *value iteration* for finding optimal policies.

Value Iteration Value iteration (Howard 1960) was developed for finding optimal policies for MDPs. Since we have formulated the partially observable problem as an MDP over belief states, we can find optimal policies for POMDPs in an analogous manner.

The agent moves through the world according to its policy, collecting reward. Although there are many criterion possible for choosing one policy over another, we here focus on policies that maximize *the infinite expected sum of discounted rewards* from all states. In such *infinite horizon* problems, we seek to maximize $E[r(0) + \sum_{t=1}^{\infty} \gamma^t r(t)]$, where $0 \leq \gamma < 1$ is a *discount factor* and $r(t)$ is the reward received at time t. If γ is zero, the agent seeks to maximize the reward for only the next time step with no regard for future consequences. As γ increases, future rewards play a larger role in the decision process.

The *optimal value* of any belief state b is the infinite expected sum of discounted rewards starting in state b and executing the optimal policy. The *value function*, $V^*(b)$, can be expressed as a system of simultaneous equations as follows:

$$V^*(b) = \max_{a \in \mathcal{A}} [\rho(b, a) + \gamma \sum_{b' \in \mathcal{B}} \tau(b, a, b') V^*(b')] \ . \quad (1)$$

The value of a state is its instantaneous reward plus the discounted value of the next state after taking the action that maximizes this value.

One could also consider a policy that maximizes reward over a finite number of time steps, t. The essence of value iteration is that optimal *t-horizon* solutions approach the optimal infinite horizon solution as t tends toward infinity. More precisely, it can be shown that the maximum difference between the value function of the optimal infinite horizon policy, V^*, and the analogously defined value function for the optimal t-horizon policy, V_t^*, goes to zero as t goes to infinity.

This property leads the following value iteration algorithm:

Let $V_0(b) = 0$ for all $b \in \mathcal{B}$
Let $t = 0$
Loop
  $t := t + 1$
  For all $b \in \mathcal{B}$
    $V_t(b) = \max_{a \in \mathcal{A}} [\rho(b, a)$
      $+ \gamma \sum_{b' \in \mathcal{B}} \tau(b, a, b') V_{t-1}(b')]$
Until $|V_t(b) - V_{t-1}(b)| < \epsilon$ for all $b \in \mathcal{B}$

This algorithm is guaranteed to converge in a finite number of iterations and results in a policy that is within $2\gamma\epsilon/(1-\gamma)$ of the optimal policy (Bellman 1957; Lovejoy 1991).

In finite state MDPs, value functions can be represented as tables. For this continuous space, however, we need to make use of special properties of the belief MDP to represent it finitely. First of all, any finite horizon value function is piecewise linear and convex (Sondik 1971; Smallwood & Sondik 1973). In addition, for the infinite horizon, the value function can be approximated arbitrarily closely by a convex piecewise-linear function (Sondik 1971).

A representation that makes use of these properties was introduced by Sondik (Sondik 1971). Let $\mathcal{V}_t$ be a set of $|\mathcal{S}|$-dimensional vectors of real numbers. The optimal t-horizon value function can be written as:

$$V_t(b) = \max_{\alpha \in \mathcal{V}_t} b \cdot \alpha \ ,$$

for some set $\mathcal{V}_t$. Any piecewise-linear convex function can be expressed this way, but the particular vectors in $\mathcal{V}_t$ can also be viewed as the values associated with different choices in the optimal policy, analogous to Watkins' Q-values (Watkins & Dayan 1992); see (Cassandra, Kaelbling, & Littman 1994; Sondik 1971).

The Witness Algorithm The task at each step in the value iteration algorithm is to find the set $\mathcal{V}_t$ that represents V_t^* given $\mathcal{V}_{t-1}$. Detailed algorithms have been developed for this problem (Smallwood & Sondik 1973; Monahan 1982; Cheng 1988) but are extremely inefficient. We describe a new algorithm, inspired by Cheng's linear support algorithm (Cheng 1988), which both in theory and in practice seems to be more efficient than the others.

Many algorithms (Smallwood & Sondik 1973; Cheng 1988) construct an approximate value function,

$\hat{V}_t(b) = \max_{\alpha \in \hat{\mathcal{V}}_t} b \cdot \alpha$, which is successively improved by adding vectors to $\hat{\mathcal{V}}_t \subseteq \mathcal{V}_t$. The set $\hat{\mathcal{V}}_t$ is built up using a key insight. From $\mathcal{V}_{t-1}$ and any particular belief state, b, we can determine the $\alpha \in \mathcal{V}_t$ that should be added to $\hat{\mathcal{V}}_t$ to make $\hat{V}_t(b) = V_t^*(b)$. The algorithmic challenge, then, is to find a b for which $\hat{V}_t(b) \neq V_t^*(b)$ or to prove that no such b exists (i.e., that the approximation is perfect).

The Witness algorithm (Cassandra, Kaelbling, & Littman 1994) defines a linear program that returns a single point that is a "witness" to the fact that $\hat{V}_t \neq V_t^*$. The process begins with an initial $\hat{\mathcal{V}}_t$ populated by the vectors needed to represent the value function at the *corners* of the belief space (i.e., the $|\mathcal{S}|$ belief states consisting of all 0's and a single 1). A linear program is constructed with $|\mathcal{S}|$ variables used to represent the components of a belief state, b. Auxiliary variables and constraints are used to define

$$v = V_t^*(b) = \max_{a \in \mathcal{A}} [\rho(b, a) + \gamma \sum_{b' \in \mathcal{B}} \tau(b, a, b') \max_{\alpha \in \mathcal{V}_{t-1}} \alpha \cdot b']$$

and

$$\hat{v} = \hat{V}_t(b) = \max_{\hat{\alpha} \in \hat{\mathcal{V}}_t} \hat{\alpha} \cdot b$$

A final constraint insists that $\hat{v} \neq v$ and thus the program either returns a witness or fails if $\hat{V}_t = V_t^*$. If a witness is found, it is used to determine a new vector to include in $\hat{\mathcal{V}}_t$ and the process repeats. Only one linear program is solved for each vector in $\hat{\mathcal{V}}_t$.

In the current formulation, a tolerance factor, δ, must be defined for the linear program to be effective. Thus the algorithm can terminate even though $V_t^* \neq \hat{V}_t$ as long as the difference at any point is no more than δ. This differentiates the Witness algorithm from the other approaches mentioned, which find exact solutions.

Although the Witness algorithm only constructs approximations, in conjunction with value iteration it can construct policies arbitrarily close to optimal by making δ small enough. Unfortunately, extremely small values of δ result in numerically unstable linear programs that can be quite challenging for many linear programming implementations.

It has been shown that finding the optimal policy for a finite-horizon POMDP is PSPACE-complete (Papadimitriou & Tsitsiklis 1987), and indeed all of the algorithms mentioned take time exponential in the problem size if the specific POMDP parameters require an exponential number of vectors to represent V_t^*. The main advantage of the Witness algorithm is that it appears to be the only one of the algorithms whose running time is guaranteed not to be exponential if the number of vectors required is not. In practice, this has resulted in vastly improved running times and the ability to run much larger example problems than existing POMDP algorithms. Details of the algorithm are outlined in a technical report (Cassandra, Kaelbling, & Littman 1994).

Representing Policies When value iteration converges, we are left with a set of vectors, $\mathcal{V}_{\text{final}}$, that constitutes an approximation to the optimal value function, V^*. Each of these vectors defines a region of the belief space such that a belief state is in a vector's region if its dot product with the vector is maximum. Thus, the vectors define a partition of belief space.

It can be shown (Smallwood & Sondik 1973) that all state vectors that share a partition also share an optimal action, so a policy can be specified by a set of pairs, $\langle \alpha^*, a^* \rangle$, where $\pi(b) = a^*$ if $\alpha^* \cdot b \geq \alpha \cdot b$ for all $\alpha \in \mathcal{V}_{\text{final}}$.

For many problems, the partitions have an important property that leads to a particularly useful representation for the optimal policy. Given the optimal action and a resulting observation, all belief states in one partition will be transformed to belief states occupying the same partition on the next step. The set of partitions and their corresponding transitions constitute a *policy graph* that summarizes the action choices of the optimal policy.

Figure 3 shows the policy graph of the optimal policy for the simple POMDP environment of Figure 2. Each node in the picture corresponds to a set of belief states over which one vector in $\mathcal{V}_{\text{final}}$ has the largest dot product and is labeled with the optimal action for that set of belief states. Observations label the arcs of the graph, specifying how incoming information affects the agent's choice of future actions. The agent's initial belief state is in the node marked with the extra arrow. In the example figure we chose the uniform belief distribution to indicate that the agent initially has no knowledge of its situation.

Using a Policy Graph Once computed, the policy graph is a representation of the optimal policy. The agent chooses the action associated with the start node, and then, depending on which observation it makes, the agent makes a transition to the appropriate node. It executes the associated action, follows the arc for the next observation, and so on.

For a reactive agent, this representation of a policy is ideal. The current node of the policy graph is sufficient to summarize the agent's past experience and its future decisions. The arcs of the graph dictate how new information in the form of observations is incorporated into the agent's decision-making. The graph itself can be executed simply and efficiently.

Returning to Figure 3, we can give a concrete demonstration of how a policy graph is used. The policy graph can be summarized as "Execute the pattern right, right, left, stopping when the goal is encountered. Execute the action left to reset. Repeat." It is straightforward to verify that this strategy is indeed optimal; no other pattern performs better.

Note that the use of the state estimator, SE, is no longer necessary for the agent to choose actions optimally. The policy graph has all the information it needs.

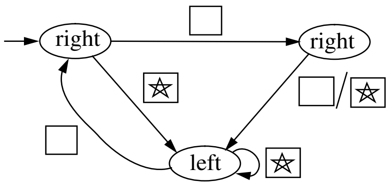

Figure 3: Sample policy graph for the simple POMDP environment

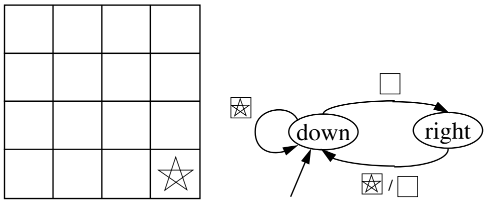

Figure 4: Small unobservable grid and its policy graph

Results

After experimenting with several algorithms for solving POMDPs, we devised and implemented the Witness algorithm and a heuristic method for constructing a policy graph from $\mathcal{V}_{\text{final}}$. Although the size of the optimal policy graph can be arbitrarily large, for most of the problems we tried the policy graph included no more than thirty nodes. The largest problem we looked at consisted of 23 states, 4 actions and 11 observations and our algorithm converged on a policy graph of four nodes in under a half of an hour.

Generalization In the policy graph of Figure 3, there is almost a one-to-one correspondence between nodes and the belief states encountered by the agent. For some environments, the decisions for many different belief states are captured in a small number of nodes. This constitutes a form of generalization in that a continuum of belief states, including distinct environmental states, are handled identically.

Figure 4 shows an extremely simple environment consisting of a 4 by 4 grid where all cells except for the goal in the lower right-hand corner are indistinguishable. The optimal policy graph for this environment consists of just two nodes, one for moving down in the grid, and the other for moving to the right. From the given start node, the agent will execute a down-right-down-right pattern until it reaches the goal at which point it will start again. Note that each time it is in the "down" node, it will have different beliefs about what environmental state it is in.

Acting to Gain Information In many real-world problems, an agent must take specific actions to gain information that will allow it to make more informed decisions and achieve increased performance. In most planning systems, these kinds of actions are handled differently than actions that change the state of the environment. A uniform treatment of actions of all kinds is desirable for simplicity, but also because there are many actions that have both material and informational consequences.

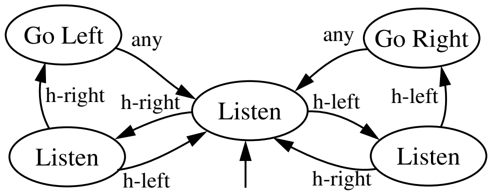

Figure 5: Policy graph for the tiger problem

To illustrate the treatment of information-gathering actions in the POMDP model, we introduce a modified version of a classic problem. You stand in front of two doors: behind one door is a tiger and behind the other is a vast reward, but you do not know which is where. You may open either door, receiving a large penalty if you chose the one with the tiger and a large reward if you chose the other. You have the additional option of simply listening. If the tiger is on the left, then with probability 0.85 you will hear the tiger on your left and with probability 0.15 you will hear it on your right; symmetrically for the case in which the tiger is on your right. If you listen, you will pay a small penalty. Finally, the problem is iterated, so immediately after you choose either of the doors, you will again be faced with the problem of choosing a door; of course, the tiger has been randomly repositioned.

The problem is this: How long should you stand and listen before you choose a door? The Witness algorithm found the solution shown in Figure 5. If you are beginning with no information, then you enter the center node, in which you listen. If you hear the tiger on your left, then you enter the lower right node, which encodes roughly "I've heard a tiger on my left once more than I've heard a tiger on my right"; if you hear the tiger on your right, then you move back to the center node, which encodes "I've heard a tiger on my left as many times as I've heard one on my right." Following this, you listen again. You continue listening until you have heard the tiger twice more on one side than the other, at which point you choose.

As the consequences of meeting a tiger are made less dire, the Witness algorithm finds strategies that listen only once before choosing, then ones that do not bother to listen at all. As the reliability of listening is made worse, strategies that listen more are found.

Related Work

There is an extensive discussion of POMDPs in the operations research literature. Surveys by Monahan (Monahan 1982) and Lovejoy (Lovejoy 1991) are good starting points. Within the AI community, several of the issues addressed here have also been examined by researchers working on reinforcement learning. Whitehead and Ballard (Whitehead & Ballard 1991) solve problems of partial observability through access to extra perceptual data. Chrisman (Chrisman 1992) and McCallum (McCallum 1993) describe algorithms for inducing a POMDP from interactions with the environment and use relatively simple approximations to the resulting optimal value function. Other relevant work in the AI community includes the work of Tan (Tan 1991) on inducing decision trees for performing low-cost identification of objects by selecting appropriate sensory tests.

Future Work

The results presented in this paper are preliminary. We intend, in the short term, to extend our algorithm to perform policy iteration, which is likely to be more efficient. We will solve larger examples including tracking and surveillance problems. In addition, we hope to extend this work in a number of directions such as applying stochastic dynamic programming (Barto, Bradtke, & Singh 1991) and function approximation to derive an optimal value function, rather than solving for it analytically. We expect that good approximate policies may be found more quickly this way. Another aim is to integrate the POMDP framework with methods such as those used by Dean *et al.* (Dean *et al.* 1993) for finding approximately optimal policies quickly by considering only small regions of the search space.

Acknowledgments

Thanks to Lonnie Chrisman, Tom Dean and (indirectly) Ross Schachter for introducing us to POMDPs.

References

Astrom, K. J. 1965. Optimal control of markov decision processes with incomplete state estimation. *J. Math. Anal. Appl.* 10:174–205.

Barto, A. G.; Bradtke, S. J.; and Singh, S. P. 1991. Real-time learning and control using asynchronous dynamic programming. Technical Report 91-57, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts.

Bellman, R. 1957. *Dynamic Programming.* Princeton, New Jersey: Princeton University Press.

Cassandra, A. R.; Kaelbling, L. P.; and Littman, M. L. 1994. Algorithms for partially observable markov decision processes. Technical Report Forthcoming, Brown University, Providence, Rhode Island.

Cheng, H.-T. 1988. *Algorithms for Partially Observable Markov Decision Processes.* Ph.D. Dissertation, University of British Columbia, British Columbia, Canada.

Chrisman, L. 1992. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 183–188. San Jose, California: AAAI Press.

Dean, T.; Kaelbling, L. P.; Kirman, J.; and Nicholson, A. 1993. Planning with deadlines in stochastic domains. In *Proceedings of the Eleventh National Conference on Artificial Intelligence.*

Howard, R. A. 1960. *Dynamic Programming and Markov Processes.* Cambridge, Massachusetts: The MIT Press.

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

McCallum, R. A. 1993. Overcoming incomplete perception with utile distinction memory. In *Proceedings of the Tenth International Conference on Machine Learning.* Amherst, Massachusetts: Morgan Kaufmann.

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

Moore, R. C. 1985. A formal theory of knowledge and action. In Hobbs, J. R., and Moore, R. C., eds., *Formal Theories of the Commonsense World.* Norwood, New Jersey: Ablex Publishing Company.

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

Smallwood, R. D., and Sondik, E. J. 1973. The optimal control of partially observable markov processes over a finite horizon. *Operations Research* 21:1071–1088.

Sondik, E. J. 1971. *The Optimal Control of Partially Observable Markov Processes.* Ph.D. Dissertation, Stanford University, Stanford, California.

Sutton, R. S. 1990. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning.* Austin, Texas: Morgan Kaufmann.

Tan, M. 1991. Cost-sensitive reinforcement learning for adaptive classification and control. In *Proceedings of the Ninth National Conference on Artificial Intelligence.*

Watkins, C. J. C. H., and Dayan, P. 1992. Q-learning. *Machine Learning* 8(3):279–292.

Whitehead, S. D., and Ballard, D. H. 1991. Learning to perceive and act by trial and error. *Machine Learning* 7(1):45–83.