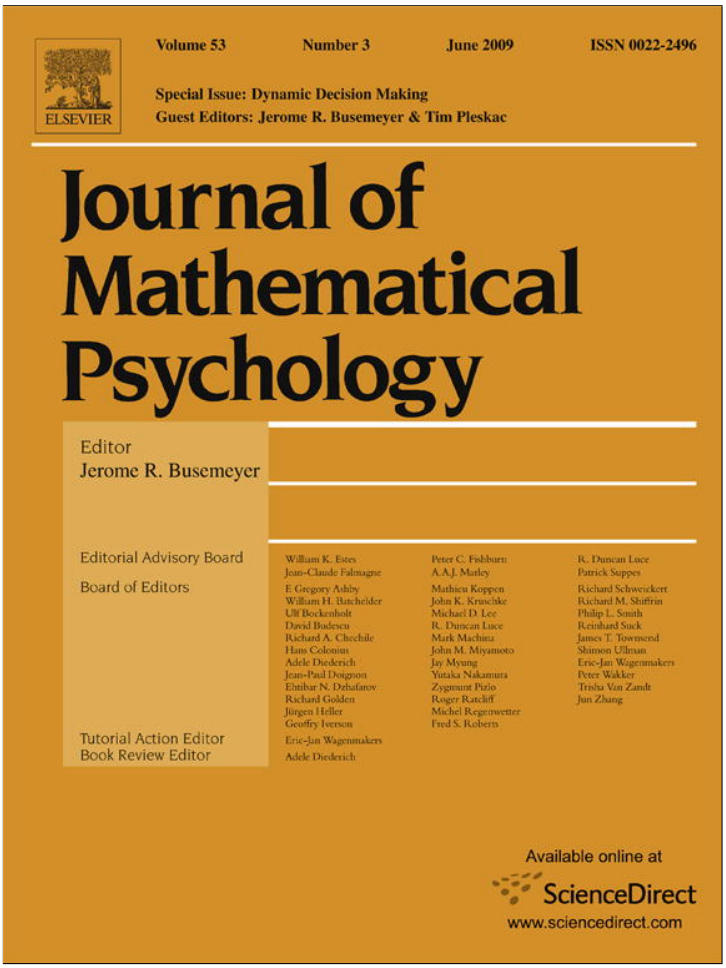

Journal of Mathematical Psychology 53 (2009) 119–125

Contents lists available at ScienceDirect

Journal of Mathematical Psychology

journal homepage: www.elsevier.com/locate/jmp

A tutorial on partially observable Markov decision processes

Michael L. Littman

Department of Computer Science, Rutgers University, Piscataway, NJ 08854-8019, USA

ARTICLE INFO

Article history:
Received 22 December 2008
Available online 11 February 2009

Keywords:
Markov processes
POMDP
Planning under uncertainty

ABSTRACT

The partially observable Markov decision process (POMDP) model of environments was first explored in the engineering and operations research communities 40 years ago. More recently, the model has been embraced by researchers in artificial intelligence and machine learning, leading to a flurry of solution algorithms that can identify optimal or near-optimal behavior in many environments represented as POMDPs. The purpose of this article is to introduce the POMDP model to behavioral scientists who may wish to apply the framework to the problem of understanding normative behavior in experimental settings. The article includes concrete examples using a publicly-available POMDP solution package.

1. Introduction

The goal of this article is to familiarize behavioral scientists with the partially observable Markov decision process (POMDP) model and some of the computational tools available for working with these models. It steps through several concrete examples using a publicly-available software package called "pomdp-solve".[1]

Both behavioral scientists and computer scientists study the interaction between decision makers or agents (whether animals, people, computers, algorithms, or organizations) and environments (whether problems, puzzles, experimental apparatuses, or ecological niches). A behavioral scientist takes the agent as *given* and designs or controls an environment to gain insight into the behavior of the agent. To a behavioral scientist, "data" is concerned with how the agent reacted to its environment. He will sometimes engage in modeling of the agent to crystalize his understanding of its dynamics. A behavioral scientist does not typically build models of environments—to the extent that the environment is of interest, it can often serve as its own model.

A computer scientist, in contrast, takes the *environment* as given. She seeks to design and deploy an agent that produces the best possible behavior for its environment. When a computer scientist collects data, it is often measuring contingencies in the environment with the goal of modeling or exploiting them for the benefit of the agent. A computer scientist does not *model* the agent, she *creates* it.

A POMDP model formalizes the interaction between agents and environments, as shown in Fig. 1. As such, it is of equal applicability in the behavioral- and computer-science settings. However, the majority of algorithmic work in the POMDP setting has been on the development of algorithms for "solving" a POMDP. A solving algorithm takes a specification of an environment and produces a specification of an agent that produces the best possible behavior in that environment. Thus, solving algorithms implement a computer-science agenda. Nevertheless, since the two agendas are really two sides of a coin, I believe an introduction to POMDP solution algorithms is valuable to behavioral scientists as well.

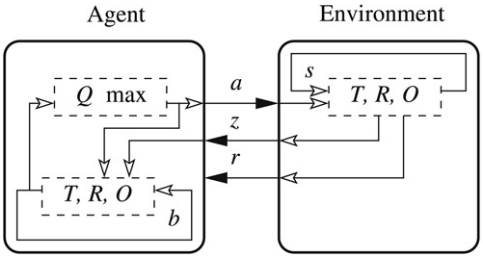

Fig. 1. The environment–agent interaction. The agent chooses action a, causing a state transition that generates an observation z and reward r. The interchange repeats after the agent chooses its next action—the figure illustrates one architecture for choosing actions, discussed in a later section.

The next section introduces the POMDP model mathematically. Section 4 provides some basics about solving POMDPs. The Appendix presents a series of example environments and their solutions. The final section concludes and mentions some current research trends.

2. The POMDP model

The POMDP model was introduced by Aström (1965) and Smallwood and Sondik (1973) provided the first general algorithmic approaches a few years later. They generalize the Markov decision process model (Bellman, 1957) by incorporating the idea

E-mail address: mlittman@cs.rutgers.edu.
URL: http://www.cs.rutgers.edu/~mlittman/.

[1] Pomdp-solve was developed by Tony Cassandra and can be found online at pomdp.org.

0022-2496/$ – see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jmp.2009.01.005

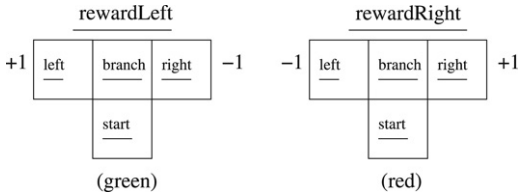

Fig. 2. An example environment ("Light") where an agent needs to observe the "color" at the start location to correctly decide which way to turn at the branch to obtain the +1 reward.

that decision makers might not be able to perfectly monitor the world state. Several excellent algorithmic surveys have been compiled (Lovejoy, 1991; Monahan, 1982; White, 1991). A more technical version of this introduction can be found elsewhere (Kaelbling, Littman, & Cassandra, 1998).

Before diving down into notation, let us begin with a concrete example of an environment we might want to model. We will call this environment "Light".

An agent is placed in a T-maze. A colored light, visible by looking up from the starting position, is green if there is a reward in the left arm of the maze and a punishment in the right arm. It is red if there is a reward in the right arm and a punishment in the left arm. After reaching the end of an arm of the maze, the subject is returned to the beginning of the maze and the position of the reward is randomized.

Producing a POMDP description of an environment requires that we specify a finite set of *states* S, *actions* A, *observations* Z, and functions detailing their relationships. One model of the Light environment is to capture the T-maze itself with a set of 4 locations. The locations are labeled as start, branch, left, and right. The actions move the agent through the maze. They are forward, left, right, and look up.

From start, forward changes the location to branch. From branch, left changes the location to left and right changes the location to right. Action forward from locations left or right terminate the trial. Actions not listed above result in no change in location.

The agents's state consists of its location (start, branch, left, or right) and the location of the reward (rewardLeft or rewardRight) for a total of 8 possible combinations. The rewards in the Light environment are all zero except for +1 for taking action forward from left–rewardLeft or right–rewardRight and −1 for taking action forward from right–rewardLeft or left–rewardRight.

We can imagine that the agent can observe its location in the maze and, when it takes action look up, it observes *green* from start–rewardLeft and *red* from start–rewardRight. Note that the fact that the agent does not simply *know* its state and must take actions to gain state-related information is a hallmark of POMDPs and the most important distinction from the better known Markov decision process model Puterman (1994).

Fig. 2 illustrates the states, transitions, and observations for the Light environment. A natural starting state for the environment is 50–50 probability on start–rewardLeft and start–rewardRight. An agent attempting to maximize its total reward in the Light environment could execute the following rule. First, look up. If the observation is *green*, go forward, left, forward. If the observation is *red*, go forward, right, forward. This rule gets the maximum possible total reward of +1 on every trial.

With a concrete example in mind, we can consider a more abstract, mathematical description of POMDPs. In addition to the S, A, and Z sets mentioned above, we also have three functions relating elements of these sets to each other.

The transition function T relates states and actions to next states. In particular, for every state $s \in S$, and $s' \in S$, and every action $a \in A$, $T(s, a, s')$ is a probability between zero and one that captures the likelihood that taking action a from s results in a transition to s'. For all s and a, $\sum_{s' \in S} T(s, a, s') = 1$; that is, T is a proper probability distribution over possible next states. If the dynamics are deterministic, there will be some s' for each s, a pair such that $T(s, a, s') = 1$ while all other states s'' will have $T(s, a, s'') = 0$.

The reward function R relates states and actions to real-value rewards. For every state $s \in S$, and $s' \in S$ and every action $a \in A$, $R(s, a, s')$ is the expected reward for taking action a from state s when the resulting state is s'. In environments with a goal state, it is natural to assign $R(s, a, s') = +1$ if s' is a goal and to assign $R(s, a, s') = 0$ (or a small negative reward) for all other combinations. In many environments, reward is independent of the resulting state or the selected action and the reward function can be stated more succinctly in these cases.

The observation function O relates states and actions to observations the agent receives. For every resulting state $s' \in S$, every action $a \in A$, and every observation $z \in Z$, $O(a, s', z)$ is the probability the agent will observe z from taking action a and transitioning to state s'. For all s' and a, $\sum_{z \in Z} O(a, s', z) = 1$; that is, O is a proper probability distribution over possible observations. If observations depend only on the state in which the agent arrives, there will be some z for each a, s' triple such that $O(a, s', z) = 1$ while all other observations z' will have $O(a, s', z') = 0$. In many environments, observations are independent of the selected action and the observation function can be stated more succinctly in these cases.

In environments with a known initial state distribution, b_0 provides information on the relatively likelihood of the environment's state. Specifically, for each $s \in S$, $b_0(s)$ is the probability that the state in which the agent begins is state s. For all s, $\sum_{s \in S} b_0(s) = 1$; that is, b_0 is a proper probability distribution over initial states. If the identity of the initial state is known with certainty, there will be some s such that $b_0(s) = 1$ while all other states s'' will have $b_0(s'') = 0$.

In the Light environment, any series of decisions from the initial state results in either termination with a total reward of +1, termination with a total reward of −1, or a non-terminal sequence of 0 rewards. Consider a variation of the environment, call it LightReward, in which there is a small reward, say +0.05, for action look up in either start–rewardLeft or start–rewardRight. Now, by sitting in the initial state and repeatedly executing the look up actions, the agent receives an unbounded series of positive rewards. If these rewards are simply summed up, the agent can get an unboundedly large total reward.

The concept of *discounting* is often introduced to mitigate this issue. That is, since an agent cannot realistically get an infinite sequence of positive rewards, rewards in the distant future are downweighted relative to rewards received in the near future. More precisely, a discount factor $0 \leq \gamma \leq 1$ controls the impact of rewards as a function of their distance in the future. A reward of r received t steps in the future counts as $\gamma^t r$. So, if $\gamma = 1$, reward values are simply summed—no discounting takes place. For values of γ close to one, future rewards decay exponentially in importance. For γ close to 0, only very near-term rewards count—distant rewards end up being decayed to near-zero values. For $\gamma = 0$, only immediate rewards matter and the future is ignored. The value γ can be thought of as the probability that a trial will be allowed to continue after each step. It also has connections to interest rates in economic theory.

In the LightReward environment, repeatedly executing look up from the start state results in total rewards of $0.05 + \gamma 0.05 + \gamma^2 0.05 + \gamma^3 0.05 + \gamma^4 0.05 + \cdots = \sum_{t=0}^{\infty} \gamma^t 0.05$. Factoring out the constant reward value, this quantity is equal to $0.05 \sum_{t=0}^{\infty} \gamma^t$. Let $X = \sum_{t=0}^{\infty} \gamma^t$. Note that, multiplying γ through the sum and rearranging terms we have $\gamma X = \gamma \sum_{t=0}^{\infty} \gamma^t = \sum_{t=0}^{\infty} \gamma \gamma^t =$

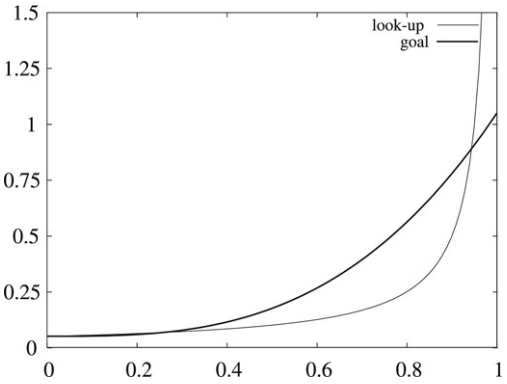

Fig. 3. Comparison of the discounted value of two strategies (look-up and goal) in the "LightReward" environment under varying discount factors. For extreme discount factors, look-up is better and for intermediate ones, goal is superior.

$\sum_{t=0}^{\infty} \gamma^{t+1} = \sum_{t=1}^{\infty} \gamma^t$. That is, it's just X without the first term $\gamma^0 = 1$. So, $\gamma X = X - 1$. Solving for X, we get $X = 1/(1-\gamma)$. That means the infinite sequence of $+0.05$ rewards has a discounted total of $0.05/(1-\gamma)$, which is finite as long as $\gamma < 1$.

Consider the impact of discounting on the LightReward environment. There are two sensible strategies in the environment—always look up ("look-up") or look up once and proceed to the correct arm ("goal"). The look-up strategy has a discounted total reward of $0.05/(1-\gamma)$, as just described. The goal strategy obtains rewards of 0.05 on the first step and $+1$ on the fourth step, meaning its discounted total is $0.05 + \gamma^3$. Fig. 3 compares the discounted total reward of these two strategies as a function of the discount factor γ. For low discount factors (0 to about 0.2599), the $+1$ reward is too distant, and look-up results in a higher total. For high discount factors (around 0.9439 and above), the discounted sum of the 0.05 reward values makes look-up result in a higher total. However, for intermediate discount rates, the agent is better off with goal. This example illustrates the impact of the choice of discount factor on optimal behavior.

3. Tracking belief states

In developing algorithms that compute optimal behavior given a specification of a POMDP environment, there are a number of important concepts. This section describes *belief states*, which allow us to encapsulate the knowledge an agent can glean about its state from interacting with the environment.

To develop this concept, let us return to the Light environment from the previous section. The environment consists of 9 states, which we can order (arbitrarily) as:

- start–rewardLeft
- branch–rewardLeft
- left–rewardLeft
- right–rewardLeft
- start–rewardRight
- branch–rewardRight
- left–rewardRight
- right–rewardRight
- done

The initial state distribution b_0 assigns a probability of 0.5 to each of the states at the starting location. We can write this function as a 9-position vector: [0.5 0.0 0.0 0.0 0.5 0.0 0.0 0.0 0.0]. This vector can be thought of as a state-occupation vector in that it captures the relative likelihood of each possible current state. It is also sometimes called a *belief state* because it summarizes the agent's knowledge of its current situation.

Let us explore what happens to the belief state after each possible action. From the two states with non-zero probability, the actions left and right have no effect so the likelihood of occupying each of the states remains unchanged—the new belief state is the same as b_0. On the other hand, the action forward moves the location from start to branch, so state start–rewardLeft transitions to branch–rewardLeft and state start–rewardRight transitions to branch–rewardRight. As a result, after the forward action, the new belief state can be written [0.0 0.5 0.0 0.0 0.0 0.5 0.0 0.0 0.0].

By similar reasoning, following this action with right will leave the agent in right–rewardLeft or right–rewardRight with equal probability. Thus, the belief state is [0.0 0.0 0.0 0.5 0.0 0.0 0.0 0.5 0.0]. The action forward will now end the trial with a transition to done and a reward of $+1$ if the state is right–rewardRight and -1 if the state is right–rewardLeft. Since these two outcomes are equally likely, the expected (average) reward is 0.

Alternatively, let us consider how the initial belief state b_0 changes under the influence of the look up action. If the state is start–rewardLeft (probability 0.5), this action will have the effect of not changing the state and providing the observation *green*. If the state is start–rewardRight (probability 0.5), the action will have the effect of not changing the state and providing the observation *red*. So, there is a 50% chance of seeing *red* and a 50% chance of seeing *green*. If *red* is observed, the underlying state must be start–rewardRight. So, even though the location of the agent has not changed, the belief state should be modified to reflect this new information. The new belief state should be [0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0].

Now, taking actions forward then right results in transitions that leave the agent in right–rewardRight, corresponding to belief state [0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0]. Again, action forward ends the trial, but now the reward for this action is $+1.0$. Thus, taking the look up action ultimately changed the payoff of the forward action by changing the agent's belief state—its knowledge of its current state. Of course, this discussion only considered the effect on the belief state of look up producing observation *red*. With 50% probability the agent will observe *green*, resulting in a belief state of [1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0].

Although belief state updates were presented above as a kind of reasoning problem, the updates have a consistent and mathematically simple form so they can be updated mechanically. The sequence of events when an action is chosen is that the current state makes a transition, based on the transition function, then an observation is selected based on the resulting state, then the state likelihoods are updated based on the received observation. So, what is the probability of observing $z \in Z$ after taking action $a \in A$ from belief state b? Since the current state is $s \in S$ with probability $b(s)$, the following state will be $s' \in S$ with probability $T(s, a, s')$ and the observation z will then be received with probability $O(a, s', z)$, the total probability of observing z is:

$$p_z = \sum_{s \in S} b(s) \sum_{s' \in S} T(s, a, s') O(a, s', z). \tag{1}$$

Now that we observed z, how do we update the belief state to a new belief state b' that reflects the fact that the agent took action a and made an observation z? The probability assigned to state s' in the new belief state should reflect the likelihood of ending up in state s' (by virtue of the likelihood of all possible initial states and their likelihood of transitioning to s') and the likelihood that the transition would produce observation z. Finally, since the belief state needs to be a valid probability distribution over states, we must normalize the entries of the belief state so they sum to one by dividing by the likelihood of the observation z:

$$b'(s') = \sum_{s \in S} b(s) T(s, a, s') O(a, s', z) / p_z. \tag{2}$$

We can also express the expected reward obtained for taking action $a \in A$ from belief state b. We simply average the reward over possible states and next states weighted by their probabilities:

$$r_a = \sum_{s \in S} \sum_{s' \in S} b(s) R(s, a, s'). \tag{3}$$

By being able to track belief states as actions are chosen and observations made, quantify the likelihood of different observations, and express the expected rewards received due to action choices, we can now reason about the expected reward for various courses of action. The next section summarizes a particular approach to using this knowledge to select actions optimally.

4. Solving POMDPs

This section describes the use of dynamic programming to compute *value functions* over belief states en route to defining optimal behavior.

First, let us consider behaving optimally in a restricted setting in the Light environment. Let us say the current belief state is b and the agent is only allowed to take one action. We want it to get the most reward it can, on average, in this one step. The expected reward for selecting action $a \in A$ from belief state b is, rewriting Eq. (3),

$$Q_1(b, a) = \sum_{s \in S} b(s) \sum_{s' \in S} R(s, a, s').$$

Thus, the expected reward for action a is a linear function of the belief state b. The best action to chose from belief state b is the one that leads to the highest value of $Q_1(b, a)$. Simple enough.

Let us define $V_1(b)$ to be the value obtained by choosing the best action as a function of b in this one-step version of the problem. It follows that $V_1(b) = \max_a Q_1(b, a)$. Since each of the $Q_1(b, a)$ functions is a linear function of b, the function $V_1(b)$ is *piecewise linear and convex*; that is, it is the maximum over a finite set of linear functions.

The $V_1(b)$ function tells us, essentially, how good it is for the agent to be in belief state b given that it has one step to gather all the reward it can. It is sometimes called the one-step value function. We can use the one-step value function to decide the best course of action to adopt given that the agent has *two* steps to gather all the reward it can.

Specifically, let us say that b is the belief state and the agent can act for two steps. If action a is chosen on the first step, the agent Eq. (3) gets an immediate reward of

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

In addition to this reward, the action always results in a transition to a new belief state b', depending on the observation that is received. This observation will be z with probability p_z defined in Eq. (1) and the resulting belief state in this case is b' defined in Eq. (2). Choosing the last action optimally from b' results in an expected reward of $V_1(b')$.

Putting these pieces together mathematically, the expected reward for choosing action a from belief state b and then making the best one-step decision from the resulting belief state is:

$$Q_2(b, a) = r_a + \gamma \sum_{z \in Z} p_z V_1(b'). \tag{4}$$

Because V_1 is piecewise linear and convex and because the p_z factor is used to normalize b' and also weight the resulting value, it can be shown that this Q_2 function is also piecewise linear and convex. Similarly, defining $V_2(b) = \max_{a \in A} Q_2(b, a)$ as the expected value for taking the best two-step strategy from belief state b, we find that this value function, too, is piecewise linear and convex.

The details of the form of these functions have been detailed elsewhere (Cassandra, Littman, & Zhang, 1997; Smallwood & Sondik, 1973; Sondik, 1971; White & Scherer, 1989) and are beyond the scope of this tutorial. However, it is important to note that the transformation of V_1 to V_2 can be applied inductively. That is, if we have the (piecewise linear and convex) value function $V_t(b)$ represented by a set of linear functions that captures the expected value of the optimal t-step strategy starting from belief state b, we can construct V_{t+1} the (piecewise linear and convex) value function that captures the expected value of the optimal $(t+1)$-step strategy starting from belief state b. This insight is the essence of exact finite-horizon POMDP algorithms.

As t grows, the complexity of $V_t(b)$ (as measured by the total number of linear functions needed to represent it) can grow fast (but not more than doubly exponentially in t). So, for some environments, solving the POMDP can be intractable for all but the smallest values of t.

Perhaps even worse, the number of linear functions can continue to grow without bound. In particular, when the optimal policy is not "finitely transient" (Sondik, 1978), an infinite number of linear functions is needed to capture the optimal value function in the infinite horizon. Although no one has been able to characterize precisely when infinite-horizon policies can be finitely represented, some algorithms search directly in the space of approximately optimal finite-state controllers to mitigate this problem (Hansen, 1997; Poupart & Boutilier, 2003).

So, whereas we would like to know how to direct the agent to behave optimally with an unbounded number of steps, we end up having to satisfy ourselves with behavior that is, at best, optimal only for a fixed window (horizon).

While the worst-case situation is rather bleak, many environments are considerably easier to work with. The Appendix steps through a few simple examples in sufficient detail that the interested reader should be able to replicate the experiments.

5. Conclusions

Partially observable Markov decision processes (POMDPs) generalize Markov decision processes (Bellman, 1957; Puterman, 1994) to environments in which sensing and state are decoupled (Aström, 1965). Such problems are extremely common in the behavioral sciences because the discovery of the unknown is such a central theme.

The literature is replete with practical and engineering problems that have been cast as POMDPs and whose solutions have provided useful insights. Cassandra (1998) provides references to many interesting examples including cost control in accounting, corporate structure internal audit timing, learning processes, teaching strategies, moving target search, fishery policies, electric distribution network troubleshooting, questionnaire design, behavioral ecology, and elevator control. More recent examples from the artificial-intelligence community include robot navigation (Simmons & Koenig, 1995), dialogue control (Roy, Pineau, & Thrun, 2000), aiding disabled people (Boger et al., 2005), trust/reputation reasoning (Regan, Cohen, & Poupart, 2005), planetary rover control (Zilberstein, Washington, Bernstein, & Mouaddib, 2002), and map learning (Murphy, 1999).

The last few years has seen a growth of interest in understanding POMDPs theoretically and empirically and in extending the model to even richer interactions. Theoretically, we now know that solving POMDPs exactly is actually impossible in the worst case—a completely general exact algorithm could be used to solve the halting problem (Madani, Hanks, & Condon, 1999). However, progress has been made in developing more robust approximation methods. Of particular note is the family of point-based methods, which have proven themselves adept at attacking POMDP models too

complex for exact approaches (Pineau, Gordon, & Thrun, 2003; Spaan & Vlassis, 2005).

Two recent threads concern environment models inspired by work on POMDPs. One, predictive state representations or PSRs (Littman, Sutton, & Singh, 2002; Singh, James, & Rudary, 2004), describes an equally expressive model for partially observable environments that eschews states and instead represents environments using predictions of future observation sequences. The goal of this work has been to develop a representation that makes the learning of environment structure more direct. The second is a distributed model of decision making, suitable for execution by coordinated teams of agents. One prominent example is that of decentralized POMDPs, or DEC-POMDPs (Bernstein, Zilberstein, & Immerman, 2000). Like POMDPs, these models provide a concrete mathematical formalism for talking about environments of interest to researchers in artificial intelligence. Although they are considerably younger than POMDPs, they have quickly attracted attention and a steady flow of empirical and theoretical results has been generated (Bernstein, Hansen, & Zilberstein, 2005).

All in all, the computational work on POMDPs has matured rapidly and a suite of tools is now available for exploring individual POMDP environments. We would love to see these tools put to use in the service of important problems in the behavioral sciences.

Appendix. Example POMDP details

In this section, I step through the process of applying the pomdp-solve code to a few simple POMDPs. I will assume some familiarity with UNIX in what follows. The POMDP issues I describe transcend the details of the operating system, but having a concrete example makes it considerably easier to avoid vacuous generalities.

For this demonstration, I obtained the code from the pomdp.org website; specifically, I did:

```
wget http://pomdp.org/pomdp/code/pomdp-solve-5.3.tar.gz
gunzip pomdp-solve-5.3.tar.gz
tar xf pomdp-solve-5.3.tar.gz
```

The pomdp-solve code seems to work well on a variety of systems. To compile it, I ran:

```
./configure
make
```

Now, we need a POMDP specification to test things out. The POMDPs I describe can be downloaded from my website: http://www.cs.rutgers.edu/~mlittman/topics/jmp07/.

The file Light.POMDP, repeated in Fig. 4, provides a specification of the Light environment described in Section 2.

The pomdp-solve documentation provides a detailed specification, but it may help to briefly outline the content of this file. After a few comments, the header of the file begins. It includes a specification of the discount factor, a list of the nine states in the environment (I changed start to startx because "start" is a reserved word in this format), a list of the four actions in the environment, and a list of the 6 observations. The "start include" specification gives the initial belief state, which is equiprobable on the two states listed.

Next, the file gives the transition probabilities (lines beginning with "T") for the four actions. To a first approximation, all actions are the identity matrix—they incur no change in state. The file then lists the exceptions—the state–action pairs that result in non-identity transitions. Here, the transitions closely follow the textual description given earlier.

After the transitions, the file lists the observation information in a set of lines marked with "O". The first block of lines basically says that the agent can see its current location for all actions. A second block provides the observations made when the look up action is chosen.

Finally, the last four lines provide the reward function (unlisted rewards are taken to be zero). These lines begin with "R" and say that the forward action at the end of the maze can provide either $+1$ or -1, depending on which T-maze the agent is in.

I copied this file to my current directory via:

```
wget http://www.cs.rutgers.edu/~mlittman/topics/
  jmp07/Light.POMDP
```

Then, I ran pomdp-solve on this file:

```
pomdp-solve-5.3/src/pomdp-solve -pomdp Light.
  POMDP
```

The solver ran for 6 iterations, or "epochs", producing representations of the value functions V_1 through V_6. After 6 steps, the solver realized that further iterations would not result in any changes—that is, $V_6 = V_7 = V_8 = \cdots$ because additional steps in this environment cannot increase the expected reward. It then wrote out two files, Light-27111.alpha and Light-27111.pg (the "27111" is essentially a random code—rerunning the algorithm will produce files with different names).

Section 4 explained that V_6 is piecewise linear and convex—it is the maximum of a set of linear functions over belief states. The file Light-27111.alpha contains the twenty-two linear functions needed to solve this problem. These linear functions are sometimes called α-vectors, hence the name of the file. Each occupies three lines in the file. The first line for each function is the action that is optimal to choose from all belief states where this linear function is larger than the others. The actions are listed as numerical indices in the order given in the Light.POMDP file ($0 =$ forward, $1 =$ left, $2 =$ right, and $3 =$ look up). The second line for each function is the coefficient of the function itself—one for each state, listed in the same order as they appear in the Light.POMDP file. The third line is blank.

We can use the vectors in Light-27111.alpha to determine the optimal action to take from any belief state. For example, from the belief state where start–rewardright start–rewardleft (the first two states listed in the file) are equally likely, we can average the first two coefficients of each function to see which is largest. Here is how we might ask which function is largest for this belief state:

```
awk '{ if (FNR % 3 == 2) {print (FNR-2)/3, $1 * .5
  + $2*.5}}' Light-27111.alpha
  | sort +1rn | head -1
```

It returns "13 0.857375", meaning the 14th function (numbered from zero) assigns a value of 0.857375 to this belief state. Note that this value is precisely what we would expect, since the optimal policy reached the $+1$ state after 3 steps. With the discount factor set to 0.95, that is $+1 \times 0.95^3 = 0.857375$. What action is the right action to take first from this belief state? We can count 14 functions down in the file, or note that:

```
awk '{ if (FNR % 3 == 1) {print (FNR-1)/3, $1}}'
  Light-27111.alpha
```

says that line number 13 is associated with Action 3 (look up, as expected).

In this example, a finite set of linear functions was sufficient to capture the optimal strategy. In some environments, no finite number is sufficient. In this case, the pomdp-solve program would have continued running until terminated by control-C. At this point, it would have written out a representation of the last complete value function it computed, which would be exact for the corresponding horizon.

The other file that is created by the pomdp-solve program, in those cases when the solution process converges, is the "pg" file,

```
# Example POMDP (Light) from
# "A Tutorial on Partially Observable Markov Decision Processes" (Littman, 2007)

discount: 0.95
values: reward
states: start-rewardright start-rewardleft branch-rewardright left-rewardright right-rewardright branch-rewardleft left-rewardleft right-rewardleft done
actions: forward left right lookup
observations: startx right left branch start-green start-red

start include: start-rewardright start-rewardleft

T : forward
identity
T : left
identity
T : right
identity
T : lookup
identity

T : forward : start-rewardright : branch-rewardright 1.0
T : left : branch-rewardright : left-rewardright 1.0
T : right : branch-rewardright : right-rewardright 1.0
T : forward : right-rewardright : done 1.0
T : forward : left-rewardright : done 1.0

T : forward : start-rewardleft : branch-rewardleft 1.0
T : left : branch-rewardleft : left-rewardleft 1.0
T : right : branch-rewardleft : right-rewardleft 1.0
T : forward : right-rewardleft : done 1.0
T : forward : left-rewardleft : done 1.0

T : forward : start-rewardright : start-rewardright 0.0
T : left : branch-rewardright : branch-rewardright 0.0
T : right : branch-rewardright : branch-rewardright 0.0
T : forward : right-rewardright : right-rewardright 0.0
T : forward : left-rewardright : left-rewardright 0.0

T : forward : start-rewardleft : start-rewardleft 0.0
T : left : branch-rewardleft : branch-rewardleft 0.0
T : right : branch-rewardleft : branch-rewardleft 0.0
T : forward : right-rewardleft : right-rewardleft 0.0
T : forward : left-rewardleft : left-rewardleft 0.0

O : * : start-rewardright : startx 1.0
O : * : branch-rewardright : branch 1.0
O : * : left-rewardright : left 1.0
O : * : right-rewardright : right 1.0
O : * : start-rewardleft : startx 1.0
O : * : branch-rewardleft : branch 1.0
O : * : left-rewardleft : left 1.0
O : * : right-rewardleft : right 1.0
O : * : done : startx 1.0

O : lookup : start-rewardleft : start-green 1.0
O : lookup : start-rewardright : start-red 1.0
O : lookup : start-rewardleft : startx 0.0
O : lookup : start-rewardright : startx 0.0

R: forward : left-rewardleft : * : * 1.0
R: forward : right-rewardleft : * : * -1.0
R: forward : left-rewardright : * : * -1.0
R: forward : right-rewardright : * : * 1.0
```

Fig. 4. A pomdp-solve specification for the Light POMDP.

Light-27111.pg in this case. The file consists of 22 lines, one for each linear function. The functions are aligned with those in the Light-27111.alpha file. Each line consists of the function number (starting from zero), the optimal action for that function, and then one number for each observation (listed in the same order as they appear in the Light.POMDP file).

The lines in this file constitute a *policy graph*. That is, they can be interpreted as a finite-state machine that chooses actions as a function of observation histories. For example, line number 13 says

```
13 3  0 1 0 1 0 21
```

This line says that, starting from this node in the graph (corresponding to the linear function that we previously found to be optimal from the initial belief state), the agent should choose action 3 (look up). Then, if it sees observations 0, 2, or 4 (*startx*, *left*, or *start-green*), the agent should next move to node 0 of the graph. If it sees observations 1 or 3 (*right* or *branch*), the agent should next move to node 1 of the graph. Finally, if it sees observation 5 (*start-red*), the agent should next move to node 21 of the graph. Although the graph has 22 nodes, only five are reachable from this starting node. Fig. 5 depicts that piece of the policy graph, as encoded by the file. The policy graph is essentially a flowchart that captures the optimal actions to take as a function of the observations made. The initial node (node 13) is marked with a heavy arrow. Each node is labeled with the optimal action to take from that node. The edges are labeled with observations that indicate which node to visit next as a function of the observation received. Although many graphs capture the same simple policy in this case, pomdp-solve returns one derived from considering all possible starting belief states, b_0.

Tracing out the choices made by the policy graph, we can see that it indeed encodes the optimal policy. It chooses look up, first. Then, if it sees *start-red*, it executes forward (and sees *branch*), right (and sees *right*), forward (and sees *startx* meaning done in this case). It repeats the last action indefinitely. However, if it sees *start-green* after the first action, it executes forward (and sees *branch*), left (and sees *left*), and forward (and sees *starts*). Again, it repeats the last action indefinitely.

I created three versions of the LightReward environment, one for a low discount factor of 0.25 (LightReward-low.POMDP), one for a medium discount factor of 0.75 (LightReward-med.POMDP), and one for a high discount factor of 0.85 (LightReward-high.POMDP).

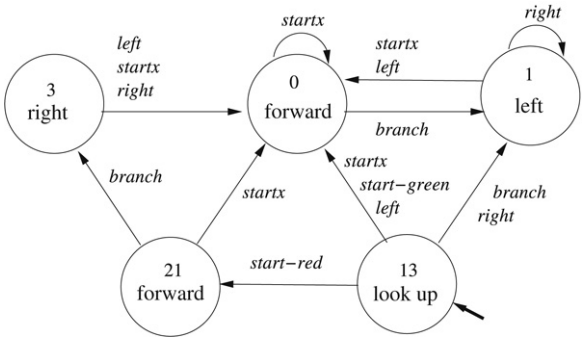

Fig. 5. The optimal policy for the Light environment expressed as a policy graph.

Otherwise, the only difference between LightReward.POMDP and Light.POMDP is the addition of two lines at the end:

```
R: lookup : * : * : * 0.05
R: lookup : done : * : * 0
```

These lines say that the look up action gives a reward of 0.05, except in the done state.

I ran pomdp-solve on the low-discount version:

```
pomdp-solve-5.3/src/pomdp-solve -pomdp
  LightReward-low.POMDP
```

and then repeated it with the other two files.

The LightReward-low.POMDP run terminated after 14 iterations and found a set of nineteen linear functions. From the initial state, all the linear functions that choose look up first have the highest expected value (0.0666667). For each of these functions, the policy graph results in a transition to node 7 independent of whether *start-green* or *start-red* is observed. Node 7 also selects look up and transitions back to itself. So, the optimal policy is to select look up on every step ($.05/(1-.25) = 0.0666667$), as predicted.

The LightReward-medium.POMDP run terminated after 69 iterations and found a set of twenty-seven linear functions. The solution from the initial state is quite similar to that of Light.POMDP.

The LightReward-high.POMDP run terminated after 433 iterations and found a solution consisting of a single linear function. The solution is to always look up no matter what the belief state. This result stands in contrast to that of the LightReward-low.POMDP environment, which always preferred look up from the initial state but found other optimal policies from other belief states. In that environment, reaching the goal is only desirable if it is nearby because the low discount factor downweights its value significantly if it is far away. On the other hand, in LightReward-high.POMDP reaching the goal results in a high reward ($+1$) and a transition to the done state where no more reward can be received. If the goal is a single step away, the total discounted reward is $+1$. On the other hand, choosing look up repeatedly results in a reward of $0.05/(1-.95) = +1$, regardless of the starting state (except for done, of course). So, always look up does no worse (and usually better) than any other strategy, regardless of starting state.

The purpose of these examples is to examine some of the issues that arise when automated POMDP solutions algorithms are applied in some simple environments. I hope they encourage you to try your own examples and explore the many rich and interesting possibilities that arise.

References

Aström, K. J. (1965). Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications*, *10*, 174–205.

Bellman, R. (1957). *Dynamic programming*. Princeton, NJ: Princeton University Press.

Bernstein, D.S., Zilberstein, S., & Immerman, N. (2000).The complexity of decentralized control of markov decision processes. In *Proceedings of the sixteenth conference on uncertainty in artificial intelligence, UAI* (pp. 32–37).

Bernstein, D.S., Hansen, E.A., & Zilberstein, S. (2005). Bounded policy iteration for decentralized POMDPs. In *Nineteenth international joint conference on artificial intelligence, IJCAI-05*.

Boger, J., Poupart, P., Hoey, J., Boutilier, C., Fernie, G., & Mihailidis, A. (2005). A decision-theoretic approach to task assistance for persons with dementia. In *Proceedings of the international joint conference on artificial intelligence, IJCAI* (pp. 1293–1299).

Cassandra, A., Littman, M. L., & Zhang, N. L. (1997). Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In *Proceedings of the thirteenth annual conference on uncertainty in artificial intelligence*, UAI–97. (pp. 54–61). San Francisco, CA: Morgan Kaufmann Publishers.

Cassandra, A.R. (1998). *Exact and approximate algorithms for partially observable markov decision problems. Ph.D. thesis*. Department of Computer Science, Brown University.

Hansen, E. A. (1997). An improved policy iteration algorithm for partially observable MDPs. *Advances in Neural Information Processing Systems*, *10*.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101*(1–2), 99–134.

Littman, M. L., Sutton, R. S., & Singh, S. (2002). Predictive representations of state. *Advances in Neural Information Processing Systems*, *14*, 1555–1561.

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

Madani, O., Hanks, S., & Condon, A. (1999). On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems. In *Proceedings of the sixteenth national conference on artificial intelligence* (pp. 541–548). The AAAI Press/The MIT Press.

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

Murphy, K. P. (1999). *Neural Information Processing Systems,*.

Pineau, J., Gordon, G., & Thrun, S. (2003). Point-based value iteration: An anytime algorithm for POMDPs. In *International joint conference on artificial intelligence, IJCAI* (pp. 1025–1032).

Poupart, P., & Boutilier, C. (2003). Bounded finite state controllers. *Advances in Neural Information Processing Systems (NIPS)*, *16*.

Puterman, M. L. (1994). *Markov decision processes—discrete stochastic dynamic programming*. New York, NY: John Wiley & Sons, Inc..

Regan, K., Cohen, R., & Poupart, P. (2005).The advisor-POMDP: A principled approach to trust through reputation in electronic markets. In *Proceedings of the third annual conference on privacy, security and trust, PST* (pp.121–130).

Roy, N., Pineau, J., & Thrun, S. (2000).Spoken dialogue management using probabilistic reasoning. In *Proceedings of the 38th annual meeting of the association for computational linguistics, ACL2000*.

Simmons, R., & Koenig, S. (1995).Probabilistic robot navigation in partially observable environments. In *Proceedings of the international joint conference on artificial intelligence* (pp. 1080–1087).

Singh, S., James, M.R., & Rudary, M.R. (2004). Predictive state representations: A new theory for modeling dynamical systems. In *Uncertainty in artificial intelligence: Proceedings of the twentieth conference, UAI* (pp. 512–519).

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

Sondik, E. (1971).*The optimal control of partially observable markov processes. Ph.D. thesis*. Stanford University.

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

Spaan, M. T. J., & Vlassis, N. (2005). Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, *24*, 195–220.

White, C. C., III, & Scherer, W. T. (1989). Solution procedures for partially observed Markov decision processes. *Operations Research*, *37*(5), 791–797.

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

Zilberstein, S., Washington, R., Bernstein, D. S., & Mouaddib, A. (2002). Decision-theoretic control of planetary rovers. In M. Beetz, et al., (Eds.), *LNAI*: *Vol. 2466*. *Plan-based control of robotic agents* (pp. 270–289).