Partially Observable Markov Decision Processes

Avi Pfeffer; Revised by David Parkes and Ryan P. Adams

In this note we return to the problem of *planning* in which we have a known probabilistic model of an uncertain environment but in which the state is not fully observable. This is formalized as the problem of *partially-observable Markov decision processes* or POMDPs. We briefly summarize some of the techniques used to solve them. The full details of POMDPs are outside the scope of this course.

1 Planning with Hidden States: POMDPs

Return again to the planning problem, where an agent has a model of its domain. But now remove the assumption that the agent always knows the state of the environment when it has to make a decision. This is much more realistic for most situations that agents encounter. The agent is not normally told exactly what the state of the world is. Rather, it gets observations about the state of the world through its *sensors*, that allow it to form probabilistic beliefs about the world state. We provide some practical examples of sensors later in these notes.

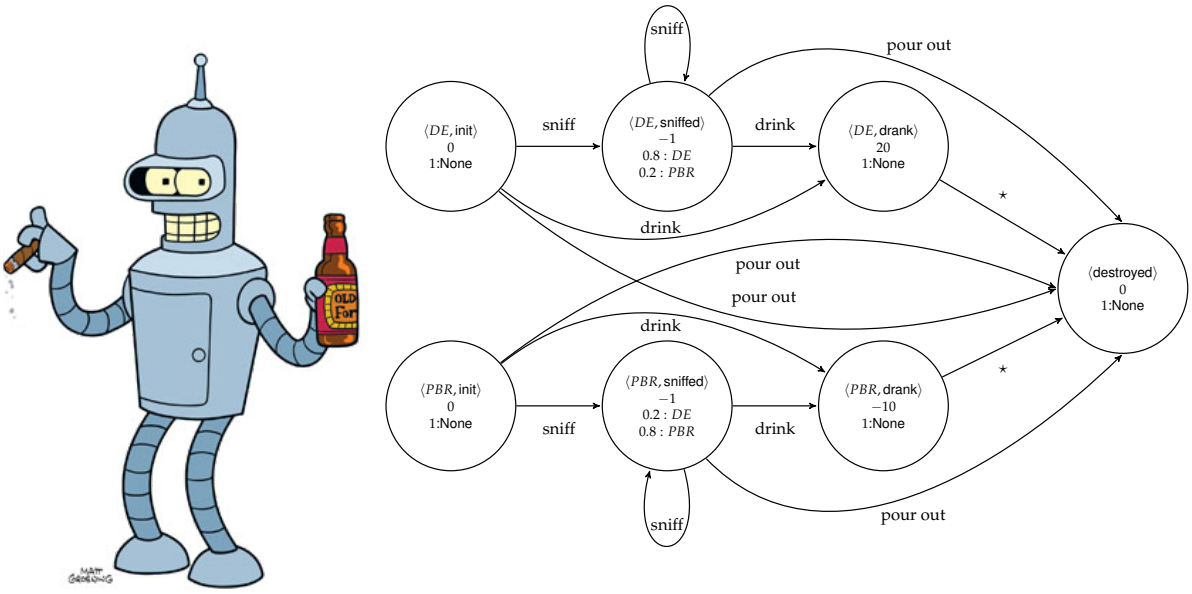
For this, we modify the Markov decision process (MDP) model to introduce noisy/incomplete **observations**. The new framework is called a *partially observable Markov decision process*, or POMDP. The overall model is similar to a blend of hidden Markov models and MDPs.

There is a set $\mathcal{O}$ of possible observations, and an *observation model* or *sensor model* that tells us what the agent is likely to observe in each state s of the world: $P(o \mid s)$ is the probability of receiving observation $o \in \mathcal{O}$ in state $s \in \mathcal{S}$. Also, we introduce an initial distribution $P^{(0)}(s)$ into the model, to model the agent's initial beliefs about states s. $P^{(0)}(s)$ is the probability that the initial state of the world is s. If there is a fixed, known, initial state s, this is modeled by setting $P^{(0)}(s) = 1$.

1.1 Example: Bender at a Party

Let's consider a simple example of an intelligent agent, Bender Bending Rodriguez (of Futurama fame), at a party. Bender's primary source of nutrition is alcohol, and he is notoriously underhanded. He is wandering around the party, drinking from unattended plastic cups. Bender is from Mexico and so his favorite beer is *Dos Equis*, but a hipster also brought a case of *PBR*, which Bender hates. When he finds an unattended beer, Bender has to decide whether to drink it or pour it out on the ground.

We can describe Bender's problem through a state transition diagram, as show in Figure 1. Bender may sniff (or "observe") an unattended drink to try to determine if it is *Dos Equis* or *PBR*. We can model this situation as a POMDP. The states determine the possible observations, so we must include enough information in the state space to specify the observation model. We will



(a) Bender Bending Rodriguez

(b) State Transition Diagram

Figure 1: (a) Our intrepid hero. (b) A state transition diagram for Bender deciding whether to drink an unattended beer, with a noisy observation model.

use two variables to describe the state space: the first, a binary variable, indicates whether or not the beer is *DE* or *PBR*. The second, a three-valued variable records the actions Bender has just taken: either he has taken no action, has sniffed the beer, or has consumed the beer. We also add a terminal state, to indicate that the interaction is over (which can be reached after drinking the beer or pouring it out). This state is simply represented by state "$\langle$destroyed$\rangle$" in the state diagram.

There are three possible observations: *None*, which is received at all times except after a sniff of the beer, *DE* and *PBR*, which are received after Bender sniffs the beer. Bender's nose is fallible: and we assume here that it has an 80% chance of providing the correct information. However, he can smell it multiple times and we assume each sniff is independent. The different observations that can be made in a state are indicated, e.g., $0.8 : DE$ and $0.2 : PBR$ in the second state at the top of the figure.

We assume a reward of $+20$ for drinking a *Dos Equis* and -10 for drinking a *PBR*. There is also a cost of -1 for sniffing the beer. The rewards/costs are indicated in the states in the state transition diagram.

The transition model is very simple in our example — in fact, the transition model is deterministic. The transition model is indicated by the arrows annotated by actions in the diagram.

To be clear: each node represents a state, while edges are labeled by actions ("drink","pour out" and "sniff"). An edge from s to s' labeled by action a means that taking action a in state s causes the world to transition to state s'. Each node contains three pieces of information: the name of the state (e.g., $\langle DE, \text{init}\rangle$ for the initial state where the beer is *Dos Equis*); the reward associated with the state; and the observation model for the state, specified as a probability distribution over observations.

The rewards here are associated with states and not (s, a) pairs. This is equivalent to having the same reward for all actions that reach a state. Not shown in the figure is $P^{(0)}$, Bender's initial belief about whether a beer is *Dos Equis* or *PBR*. If Bender decides, prior to sniffing the beer, that the probability of the beer being *Dos Equis* is p, then $P^{(0)}$ will assign probability p to state $\langle DE, \mathsf{init} \rangle$ and $1 - p$ to state $\langle PBR, \mathsf{init} \rangle$.

Note carefully that *Bender does not know* whether or not the beer is *Dos Equis* or *PBR* and thus does not know which state he is in!

1.2 Challenges of POMDPs

Why are POMDPs hard? There are two main reasons. The first reason is that the optimal action at any given point in time depends not only on the agent's current observation but on the history of observations and actions it has taken so far.

In our example, suppose Bender has just used the *sniff* action and wants to decide what to do next. The answer depends not only on the result of the sniff, but on the whole history of test results. This situation is in contrast to MDPs. Because the current state fully determines the expected current and future utility for each policy, and in an MDP the agent knows the current state, the agent can ignore history in choosing its policy. However, in POMDPs, where the agent does not know the current state, the history is relevant to determining what the current state is likely to be. Therefore, the agent may need to compute a separate policy for every possible history, which can be prohibitively expensive.

The second reason POMDPs are hard is that in making its decisions, an agent needs to take into account not only the effects of its actions on the environment, but also on its own future beliefs. It is possible that the agent needs to perform an action that is not optimal in any (physical) state, just to gather the information it needs to make decisions in the future. Such actions are called *information-gathering actions*. In our example, the sniff action is an information-gathering action. It is not optimal in any state. If the beer is *Dos Equis*, the optimal action is to drink it. If the beer is *PBR*, Bender should pour it out rather than drink it. Basically, if the agent knows what the state is, it should take the action appropriate to that state, rather than wasting energy on gathering observations. However, if the agent is sufficiently uncertain about what the state is, it should perform the information-gathering action in order to find out what it is.

Because of these issues, solving POMDPs to obtain optimal policies is extremely hard. In the finite horizon case, the cost is exponential in the horizon. In the infinite horizon case, the problem of whether or not there exists a policy that achieves a certain value is undecidable! [Madani, Hanks & Condon, AAAI99] Nevertheless we need to worry about solving POMDPs because the real world is a POMDP! In what follows, we first describe an optimal approach that makes an interesting observation about an equivalence with a fully-observable belief-state MDP and then briefly survey some approximate methods for solving POMDPs.

1.3 Belief State Methods

At any point in time, the agent can maintain a probability distribution over the current state of the world. This distribution is called its *belief state*, which is a very useful notion in thinking about POMDPs. The belief state allows us to formulate an agent's policy directly as a MDP on its belief state!

Instead of implementing a policy as a function from histories or internal states to actions, a policy is implemented by dividing the belief state into regions, and specifying an action in each region. A belief state policy in our example might be as follows:

$$\begin{array}{lll} \text{If} & P(\text{DE}) > 0.9, & \text{drink} \\ \text{If} & 0.2 < P(\text{DE}) \leq 0.9 & \text{sniff} \\ \text{If} & P(\text{DE}) \leq 0.2, & \text{pour out} \end{array}$$

An agent that implements a belief state policy needs to maintain its current belief state. At any point in time, the agent must know the current distribution over possible states, given its history of observations and actions taken. That is, at time $t+1$ it must compute

$$b(s_t) = P(s_t \mid o_1, \ldots, o_t, a_1, \ldots, a_{t-1}).$$

We denote its belief state b, representing its belief about the probability on each possible world state.

We can use the idea of filtering from hidden Markov models (which we'll study in more detail later) to track the belief state. We note that

$$\begin{aligned} b(s_t) &= P(s_t \mid o_1, \ldots, o_t, a_1, \ldots, a_{t-1}) \\ &\propto P(o_t \mid o_1, \ldots, o_{t-1}, a_1, \ldots, a_{t-1}, s_t) P(s_t \mid o_1, \ldots, o_{t-1}, a_1, \ldots, a_{t-1}) \\ &= P(o_t \mid s_t) \sum_{s_{t-1}} P(s_t, s_{t-1} \mid o_1, \ldots, o_{t-1}, a_1, \ldots, a_{t-1}) \\ &= P(o_t \mid s_t) \sum_{s_{t-1}} P(s_t \mid s_{t-1}, a_{t-1}) P(s_{t-1} \mid o_1, \ldots, o_{t-1}, a_1, \ldots, a_{t-1}) \\ &= P(o_t \mid s_t) \sum_{s_{t-1}} P(s_t \mid s_{t-1}, a_{t-1}) b(s_{t-1}) \end{aligned}$$

So, we can maintain the belief state from the observation o_t and the previous belief state, using the model $P(s_t \mid s_{t-1}, a_{t-1})$ and $P(o_t \mid s_t)$. This is our transition function for the equivalent belief state MDP! Finally, we can construct reward function

$$\hat{R}(b) = \sum_s b(s) R(s)\,.$$

Taken together, we have $\hat{R}$ and $P(b' \mid b, a)$ defining a fully observable MDP on the space of belief states. **The optimal policy for this fully observable MDP is also an optimal policy for the original POMDP.**

This is great except that the belief state is (usually) high dimensional and (always) continuous! We cannot just use the same value iteration method from before without a way to represent value functions in this high dimensional space.

Three kinds of approaches are used to solve POMDPs:

Piecewise Linear Convex The value function after any finite number of Bellman iterations can be represented by a finite collection of piecewise linear value functions on belief space, where the value function is the maximum over the set of value functions. With this observation, along with *iterative pruning* to remove value functions for subpolicies that are dominated, then exact policies can be found for 10's of underlying physical states.

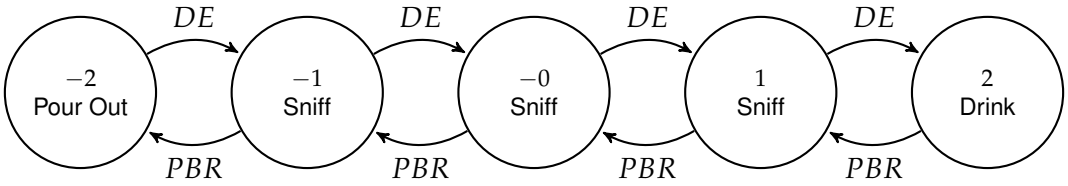


Figure 2: Finite state controller with observations and actions.

Point approximation The value function after each successive Bellman iteration is approximated via a set of point belief estimates, with interpolation methods used to evaluate the value function for any other belief.

Function approximation The value function is represented as a linear weighted sum over a set of basis functions, which are feature vectors for the set of possible sequences of observations.

1.4 Approximate Policy Iteration Methods

Since finding an optimal policy is too hard for all but the most trivial of domains, it is typical to fall back on approximate approaches. One way to achieve this is by considering a constrained policy space. *This kind of approach could be of interest in your class project.* Here are two particular kinds of policies that are often implemented:

Finite memory policy An agent with a finite memory can only remember the last K observations. So a finite memory policy is a function from the previous K percepts to an action. A typical policy for our example is to continue sniffing the beer until two identical observations are obtained in succession, at which point Bender should drink it if the observations where *Dos Equis* and pour it out if they where *PBR*.

Finite state controller The policy can be represented as a finite state controller. The nodes of the controller do not correspond to states of the world; rather, they correspond to internal states of the agent, depending on the observations it has received. In the finite state machine describing an agent controller, nodes are labeled by the action to take in that node, while edges are labeled by observations.

In the finite state controller shown in Figure 2, the nodes are labeled with an integer representing the number of *DE* sniff observations minus the number of *PBR* observations. The agent starts in state 0. Note that this is not exactly the same policy as the finite memory policy described earlier. If the percept sequence is PBR, DE, DE, the finite memory policy will say to drink, but the finite state controller will still require another sniff.

1.5 Kinds of sensors

Key to a POMDP formulation is the sensor model $P(o \mid s)$, i.e., the probability of an observation as it depends on the current world state. A sensor model is typically known (because it can be built

offline before deploying a robot in its environment). The model depends on the kind of sensor. Here are just a few examples of possible sensors.

Camera Perceptual inputs from cameras may be interpreted as single individual images, or a video stream. Also, an agent may have a single camera giving it one visual perspective on the world, or it may use multiple cameras at different locations. The visual image is normally described as a pixel array. It may be a grey-scale image, containing a light intensity (normally between 0 and 255) at every point, or it may be a color image, with RGB intensities.

Touch Many robots come with touch sensors, so that they know when they are in contact with an obstacle or a wall. Robots that manipulate objects need touch sensors to know just where and what force to apply to an object. Surprisingly, touch is also useful for robot navigation. For example, it is sometimes quite hard to control a robot's motion precisely enough so that it can head for a door and go through it without mishap. A simple solution is for the robot to aim for the wall somewhere near the door, and then slide along the wall until it reaches the door. This sliding kind of motion is called compliant motion.

Sonar Sonar is a common sensor for mobile robots. It works by emitting a sound pulse in a certain direction, and timing how long it takes for the sound to be reflected from an object. A mobile robot will typically be equipped with a circular array of eight sonars. The sonar information will tell it the distance to the nearest object in each of eight directions. Sonar only gives a robot a very rough idea of its surroundings. Furthermore, it is quite a noisy sensor. One problem is that surfaces that are not smooth do not reflect sound back directly to the robot. Nevertheless, sonar is very useful for obstacle avoidance.

Doppler radar A Doppler radar is similar to sonar in that it emits radio waves and studies how they are reflected. In addition, it measures the change in frequency between the emitted waves and the reflected waves. By the Doppler effect, if a wave is reflected by an object moving away from the radar, its frequency will decrease, while if the object is moving closer the frequency will increase. A Doppler radar is therefore able to detect motion as well as position of objects. Nevertheless, it is still limited: it can detect radial motion, toward and away from the radar, but not lateral motion. An AI problem that uses a Doppler radar is as follows: given a profile of vehicle motions over time, try to determine which vehicle went where, and perhaps what kinds of vehicles they are.

Blood Test Another "sensor" in our wider sense is a blood test. Here, the true state is the actual condition of the patient. The blood test is a noisy sensor providing information about the patient. An AI problem that uses blood test as a sensor is automated medical diagnosis.