Chapter 6

Partially observable Markov decision problems

This chapter introduces partially observable Markov decision problems*, in which an agent is faced with a sequential decision problem having only a partial view of the process state. We revisit the concept of optimality for this new class of models and derive an MDP-equivalent to POMDPs. From this equivalent MDP we port MDP solution methods to POMDPs and discuss the computation implications of this equivalence. Finally, we go over a number of approximate algorithms for POMDPs.*

6.1 Decision making under partial observability

We start with the already familiar household robot example.

Example 6.1 Consider once again the situation of a household robot that departs from the Hall after receiving a request for assistance in the Kitchen. Assuming that the location of the robot can be perfectly perceived, the decision process of the robot can be modeled as a Markov decision problem $(\mathcal{X}, \mathcal{A}, \{\boldsymbol{P}_a\}, c, \gamma)$, where

- The state space, $\mathcal{X}$, corresponds to the possible locations of the robot, i.e., $\mathcal{X} = \{B, D, H, H_1, H_2, K, L, P, W\}$.
- The action space, as in Example 5.1, is $\mathcal{A} = \{U, D, L, R, S\}$, where U (*Up*), D (*Down*), L (*Left*) and R (*Right*) move the robot to the contiguous room in the corresponding direction, and S is the action *Stay*.
- The transition probabilities include the uncertainty in certain transitions, due to steps in the environment. For example, the transitions from the Living Room to Hallway 1 fails with a probability 0.4, leaving the robot position unchanged.

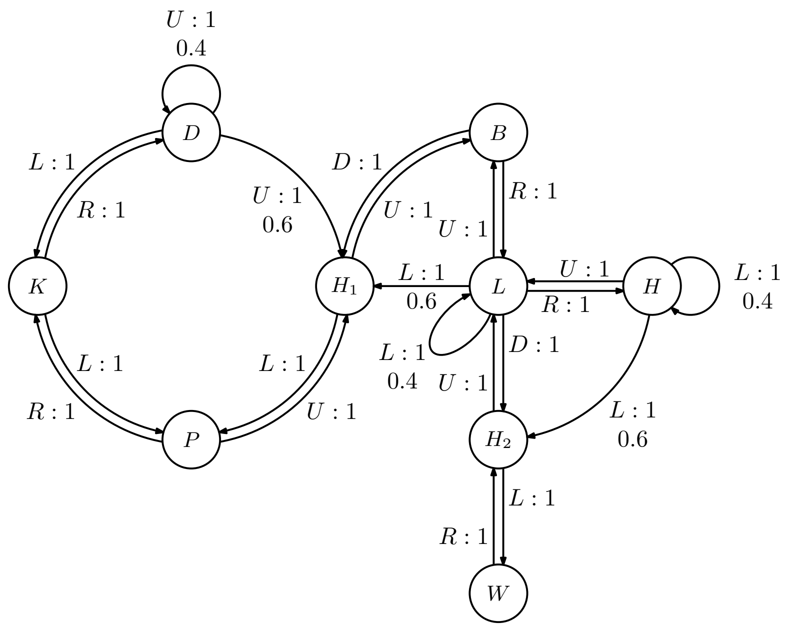


Figure 6.1 Transition diagram for the household robot Markov decision problem.

- Finally, for every time step that the robot is away from the Kitchen, it incurs a cost of 1.

The resulting MDP is represented in Fig. 6.1, where we omitted any actions that leave the state unchanged. Using any of the solvers discussed in Chapter 5, we immediately get the optimal policy depicted in Fig. 6.2.

The policy in Fig. 6.2 prescribes an action for each situation that the robot may encounter. For example, when the robot is in the Living Room, it should select action L (*Left*). To follow the policy, then, the robot must be able to determine its current situation.

Let us now suppose that the robot uses its onboard sensors to determine its current situation. In particular, as in Chapter 3, we assume that the robot is equipped with a laser sensor that detects the walls in the robot's current location. Walls with doors/passages are treated as "non-walls". The resulting pattern in each of the rooms is thus

- The Bedroom has walls on the top (t) and on the left (l);
- The Dining Room has walls on the bottom (b) and on the right (r);
- The Hallway has walls on the top (t), right (r) and bottom (b);
- Hallway 1 has no walls ($\emptyset$);

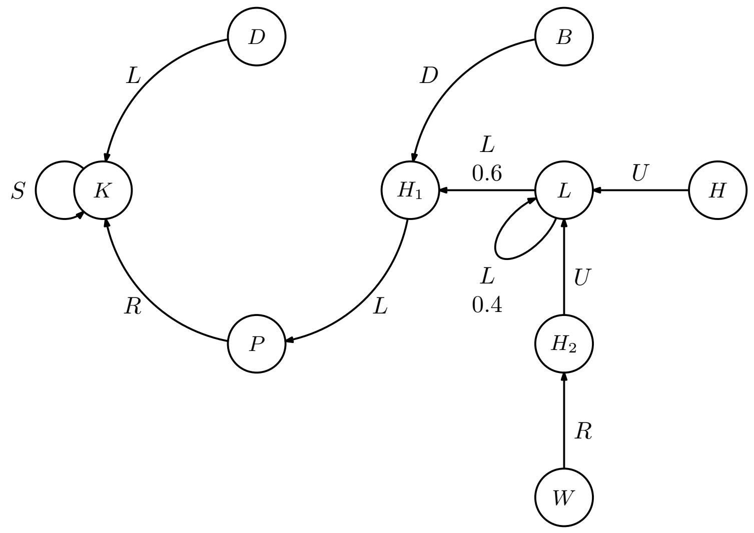


Figure 6.2 Optimal policy for the household robot problem.

- Hallway 2 has a wall at the bottom (b);
- The Kitchen has walls both at the top (t) and bottom (b);
- The Living Room as a single wall at the top (t);
- Finally, the Pantry and the WC have walls on the top (t), bottom (b) and left (l).

Except for the Pantry and the WC, the pattern of walls allows to unambiguously identify all other rooms. However, the detection is not perfect. Sometimes, the laser will fail to detect a wall when there is one, or detect a wall where there is none. Generally, when facing a wall, the laser will fail to detect it with a 5% probability. Conversely, when facing a passage, the laser will detect a wall with 10% probability, except in the left passage in the Kitchen, the top passage in Hallway 1 and the bottom passages in Hallway 1 and the Living Room, where the laser detects a wall with a 20% probability.

Suppose then that, at some time step t, the robot detects a single bottom wall. While such observation suggests that the robot may be in Hallway 2, the robot should nevertheless consider the possibility that it is, indeed, in Hallway 1, but the laser (wrongly) detected a wall at the bottom. Or that the robot is in the Dining Room but the right wall was not detected. And so on.

To make sure that the correct decision is made, one possibility is to determine the probability of each state $x \in \mathcal{X}$ from the observation and then use the expected utility theory discussed in Chapter 4. Using Bayes rule (see Appendix A),

$$\mathbb{P}\left[\mathrm{x}_t = x \mid \mathrm{z}_t = b\right] = \frac{\mathbb{P}\left[\mathrm{z}_t = b \mid \mathrm{x}_t = x\right]\mathbb{P}\left[\mathrm{x}_t = x\right]}{\mathbb{P}\left[\mathrm{z}_t = b\right]},$$

where the r.v. z_t denotes the (random) observation at time step t. Without any additional information, there is no reason to believe that one state is more likely than the other, so we set $\mathbb{P}\left[\mathrm{x}_t = x\right] = 1/\left|\mathcal{X}\right|$ to get

$$\mathbb{P}\left[\mathrm{x}_t = x \mid \mathrm{z}_t = b\right] = \frac{\mathbb{P}\left[\mathrm{z}_t = b \mid \mathrm{x}_t = x\right]}{\sum_{x' \in \mathcal{X}} \mathbb{P}\left[\mathrm{z}_t = b \mid \mathrm{x}_t = x'\right]},$$

which approximately yields the distribution

$$\boldsymbol{\beta} = \begin{bmatrix} 0.00 & 0.05 & 0.00 & 0.14 & 0.76 & 0.05 & 0.00 & 0.00 & 0.00 \end{bmatrix}.$$

As expected, the most likely state is Hallway 2. Considering the (discounted) cost-to-go resulting from each relevant state-action pair leads to the decision tree in Fig. 6.3. Finally, using the expected utility theory yields, for the different actions, the following values:

$$\begin{aligned} Q(U) &= 3.92 \\ Q(D) &= 4.39 \\ Q(L) &= 4.24 \\ Q(R) &= 4.99 \\ Q(S) &= 4.38, \end{aligned}$$

thus indicating that the best action is to move up. Unsurprisingly, U is also the action prescribed by the MDP policy for state H_2 (see Fig. 6.2).

Example 6.1 features a decision process $\left\{\left((\mathrm{x}_t, \mathrm{z}_t), \mathrm{a}_t, \mathrm{c}_t\right), t \in \mathbb{N}\right\}$, where the state of the process is a pair $(\mathrm{x}_t, \mathrm{z}_t)$; x_t corresponds to the position of the robot at time step t and z_t corresponds to the perceived wall pattern. The decision process is *partially observable*— the agent can only observe one component of the pair $(\mathrm{x}_t, \mathrm{z}_t)$, namely z_t. Therefore, from an agent's perspective, the decision process appears only as $\{(\mathrm{z}_t, \mathrm{a}_t, \mathrm{c}_t), t \in \mathbb{N}\}$ and is not Markov since, in general,

$$\mathbb{P}\left[\mathrm{z}_{t+1} = z' \mid \mathrm{h}_t = h\right] \neq \mathbb{P}\left[\mathrm{z}_{t+1} = z' \mid \mathrm{z}_t = z, \mathrm{a}_t = a\right],$$

where the history h_t is the r.v.

$$\mathrm{h}_t = \{\mathrm{z}_0, \mathrm{a}_0, \ldots, \mathrm{z}_{t-1}, \mathrm{a}_{t-1}, \mathrm{z}_t\}.$$

To distinguish between the two components x_t and z_t, we refer to x_t as the *hidden state* and to z_t as the *observation*.

Example 6.1 illustrates the impact of *partial observability* in terms of decision making. We previously discussed partial observability when we introduced HMMs in Chapter 3. Our focus then was on *prediction*—how to use the observations to

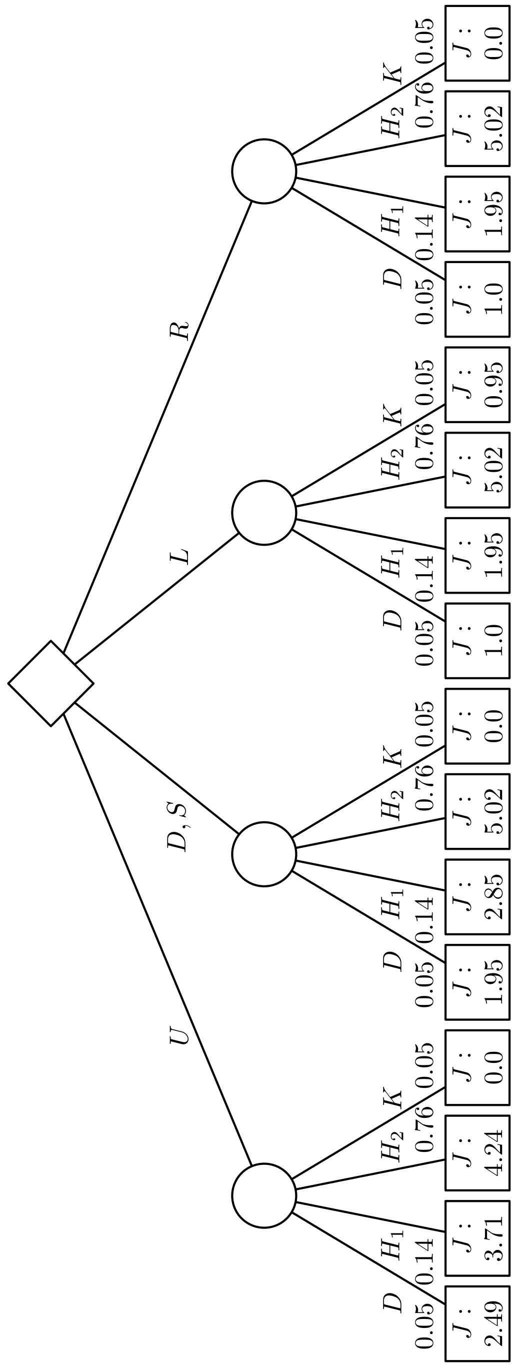


Figure 6.3 Decision tree for the situation in which the robot detects a single wall at the bottom. The values in the outcome boxes represent the cost-to-go associated with each state-action pair.

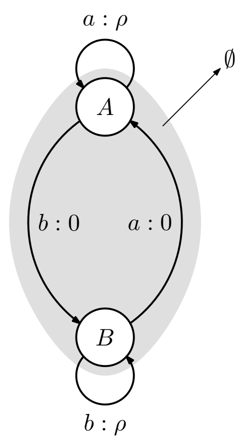


Figure 6.4 2-state MDP in which ignoring partial observability may lead to arbitrarily poor performance. The shaded area denotes the fact that the agent makes the same observation ($\emptyset$) in both states A and B.

make predictions regarding $\{\mathrm{x}_t, t \in \mathbb{N}\}$. Our focus now is on *control*—how to use the observations to make decisions towards some goal.

However, approaching control from a computational perspective should still rely on state estimation. To understand why, let us for a moment consider the alternative, where we ignore the hidden state and formulate the decision problem in terms of observations only. The resulting process, $\{(\mathrm{z}_t, \mathrm{a}_t, \mathrm{c}_t), t \in \mathbb{N}\}$, is not Markov, and is significantly less amenable to a computational treatment, as the following example establishes.

Example 6.2 Consider the decision problem depicted in Fig. 6.4 where, for the sake of argument, we allow the constant ρ to take any finite non-negative value. This problem can be represented as an MDP $(\mathcal{X}, \mathcal{A}, \{\boldsymbol{P}_a\}, c, \gamma)$, where:

- $\mathcal{X} = \{A, B\}$.
- $\mathcal{A} = \{a, b\}$.
- The transition probabilities are

$$\boldsymbol{P}_a = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \qquad\qquad \boldsymbol{P}_b = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}.$$

- The cost function can be represented in matrix form as $\boldsymbol{C} = \rho \boldsymbol{I}$, where $\boldsymbol{I}$ is the identity matrix.

Suppose now that, as in Example 6.1, the agent cannot observe the state of the MDP. Instead, at every time step t, the agent always makes the same single observation, $\mathrm{z}_t = \emptyset$, independently of the underlying (hidden) state. Therefore, the observation z_t conveys no information on x_t.

If we ignore the fact that there is a hidden state, the decision process is just $\{(\mathrm{z}_t, \mathrm{a}_t, \mathrm{c}_t), t \in \mathbb{N}\}$. The history of the process up to time step t is the r.v.

$$\mathrm{h}_t = \{\mathrm{z}_0, \mathrm{a}_0, \dots, \mathrm{z}_{t-1}, \mathrm{a}_{t-1}, \mathrm{z}_t\}$$

taking values in the set $\mathcal{H}$ of all finite histories, and a policy is any mapping $\pi : \mathcal{H} \to \Delta(\mathcal{A})$.

Let us analyze the performance of different classes of policies. Since $\mathrm{z}_t = \emptyset$ for all $t \in \mathbb{N}$, a deterministic stationary policy is necessarily constant. There are only two such policies, corresponding to always selecting action a or always selecting action b. The discounted cost-to-go associated with any such policy is, at best,

$$J = 0 + \sum_{t=1}^{\infty} \gamma^t \rho = \frac{\gamma}{1-\gamma}\rho.$$

In the underlying MDP, however, the optimal policy consists in selecting action a in state B and action b in state A, with a discounted cost-to-go of $J^* = 0$. The difference between the two is $\frac{\gamma}{1-\gamma}\rho$, which can be made arbitrarily large by increasing ρ.

Similarly, stochastic stationary policies are also necessarily constant, but may randomize between actions a and b. For example, the policy that selects actions a and b with equal probability incurs an expected cost of $\rho/2$ at every time step, yielding a discounted cost-to-go

$$J' = \sum_{t=0}^{\infty} \gamma^t \frac{\rho}{2} = \frac{1}{1-\gamma}\frac{\rho}{2}.$$

We have that

$$J - J' = \frac{\gamma}{1-\gamma}\rho - \frac{1}{1-\gamma}\frac{\rho}{2} = \frac{2\gamma - 1}{2(1-\gamma)}\rho,$$

which can be made arbitrarily large by increasing ρ when $\gamma > \frac{1}{2}$.

Finally, consider the non-stationary policy that selects action a whenever $|\mathrm{h}_t|$ is even and action b, otherwise. The discounted cost-to-go associated with this policy is, in the worst case,

$$J'' = \rho + \sum_{t=1}^{\infty} \gamma^t \times 0 = \rho.$$

Thus,

$$J - J'' = \frac{2\gamma - 1}{1-\gamma}\rho \qquad \text{and} \qquad J' - J'' = \frac{2\gamma - 1}{2(1-\gamma)}\rho,$$

both of which can be made arbitrarily large by increasing ρ.

From Example 6.2 we can conclude three fundamental facts. Let $\hat{\mathcal{M}}$ denote the process $\{(\mathrm{z}_t, \mathrm{a}_t, \mathrm{c}_t), t \in \mathbb{N}\}$, obtained from a Markov decision process $\mathcal{M} = \big\{\big((\mathrm{x}_t, \mathrm{z}_t), \mathrm{a}_t, \mathrm{c}_t\big), t \in \mathbb{N}\big\}$ by ignoring x_t. In both $\mathcal{M}$ and $\hat{\mathcal{M}}$ we consider the discounted cost-to-go as optimality criterion. Then,

(1) The optimal policy in $\hat{\mathcal{M}}$ can be arbitrarily worse than the optimal policy in $\mathcal{M}$.

(2) In $\hat{\mathcal{M}}$, the best deterministic stationary policy can be arbitrarily worse than the best stochastic stationary policy.

(3) In $\mathcal{M}$, the optimal policy may be non-stationary.

The three facts above carry important implications. First, if we ignore the hidden state, the results on the existence of deterministic and stationary optimal policies derived in Chapter 5 are no longer valid. In fact, we can not even guarantee that an optimal policy always exists for this class of decision problems.

Second, even if an optimal policy does exist, it may be non-stationary. Therefore, solution methods must search the large space of non-stationary policies, making the problem computationally much harder.

As we show in the continuation, state estimation can be used to effectively extend the main results in Chapter 5 to decision problems with partial observability, providing a definitive answer to the problem of existence of optimal policies in such domains.

6.2 Partially observable MDPs

We can now formalize decision making in the presence of partial observability.

Partially observable Markov decision process

A *partially observable Markov decision process* is sequential decision process $\mathcal{M} = \big\{\big((\mathrm{x}_t, \mathrm{z}_t), \mathrm{a}_t, \mathrm{c}_t\big), t \in \mathbb{N}\big\}$ such that

- Each r.v. x_t takes values in some finite set $\mathcal{X}$. We refer to x_t as the *hidden state* and to the set $\mathcal{X}$ as the *state space*. Moreover,

$$\mathbb{P}\left[\mathrm{x}_{t+1} = y \mid \mathbf{x}_{0:t} = \boldsymbol{x}_{0:t}, \mathbf{z}_{0:t} = \boldsymbol{z}_{0:t}, \mathbf{a}_{0:t} = \boldsymbol{a}_{0:t}\right] = \mathbb{P}\left[\mathrm{x}_{t+1} = y \mid \mathrm{x}_t = x_t, \mathrm{a}_t = a_t\right], \quad (6.1)$$

  i.e., the hidden state x_{t+1} is fully determined by x_t and a_t and independent of $\mathbf{z}_{0:t}$.

- Each r.v. z_t takes values in some finite set $\mathcal{Z}$. We refer to z_t as the *observation* and to the set $\mathcal{Z}$ as the *observation space*. Moreover,

$$\begin{aligned}\mathbb{P}\left[\mathrm{z}_{t+1} = z \mid \mathbf{x}_{0:t+1} = \boldsymbol{x}_{0:t+1}, \mathbf{z}_{0:t} = \boldsymbol{z}_{0:t}, \mathbf{a}_{0:t} = \boldsymbol{a}_{0:t}\right] \\ = \mathbb{P}\left[\mathrm{z}_{t+1} = z \mid \mathrm{x}_{t+1} = x_{t+1}, \mathrm{a}_t = a_t\right], \quad (6.2)\end{aligned}$$

i.e., the observation z_{t+1} is fully determined by the state x_{t+1} and the action a_t.

- Each r.v. a_t takes values in some finite set $\mathcal{A}$. We refer to a_t as the *action* and to the set $\mathcal{A}$ as the *action space*. The action a_t is selected by the decision maker at each time step t.
- Each r.v. c_t takes values in $[0, 1]$. We refer to c_t as the *immediate cost* at time step t. Moreover, there is a function $c : \mathcal{X} \times \mathcal{A} \to [0, 1]$ such that
$$\mathrm{c}_t = c(\mathrm{x}_t, \mathrm{a}_t), \quad (6.3)$$
i.e., c_t is a function of the state x_t and the action a_t.

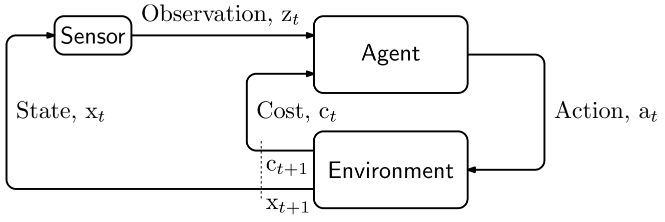


Figure 6.5 Diagram representing the interaction between the agent and the environment in a partially observable Markov decision process. In general, the observation also depends on the previous action, although we have not represented such dependence explicitly in the diagram.

The interaction between the agent and the environment in a partially observable Markov decision process is summarized in Fig. 6.5. At each time step t the agent selects an action, a_t. Depending on that action and the hidden state of the system, x_t, the agent incurs a cost $c(\mathrm{x}_t, \mathrm{a}_t)$. The environment transitions to some new state x_{t+1}, governed by transition probabilities that depend only on x_t and a_t and, as a result of such transition, the agent makes an observation z_{t+1} that depends on the current state, x_{t+1}, and on the last action, a_t. The process then repeats.

Mirroring the notation in Chapter 5, given a partially observable Markov decision process we define, for each action $a \in \mathcal{A}$, the *transition probability matrix* $\boldsymbol{P}_a$ as

$$[\boldsymbol{P}_a]_{x,y} \stackrel{\text{def}}{=} \mathbb{P}\left[\mathrm{x}_{t+1} = y \mid \mathrm{x}_t = x, \mathrm{a}_t = a\right],$$

with $x, y \in \mathcal{X}$. Similarly, we define the *observation probability matrix* $\boldsymbol{O}_a$ as

$$[\boldsymbol{O}_a]_{x,z} \stackrel{\text{def}}{=} \mathbb{P}\left[\mathrm{z}_{t+1} = z \mid \mathrm{x}_{t+1} = x, \mathrm{a}_t = a\right],$$

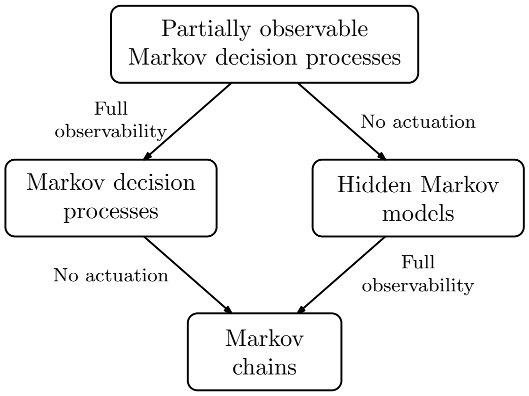


Figure 6.6 Hierarchy of Markov models.

for all $z \in \mathcal{Z}$, $x \in \mathcal{X}$ and $a \in \mathcal{A}$. We interchangeably write $\boldsymbol{P}_a(y \mid x)$ or $\boldsymbol{P}(y \mid x, a)$ to denote the element xy of matrix $\boldsymbol{P}_a$; similarly, we write $\boldsymbol{O}_a(z \mid x)$ or $\boldsymbol{O}(z \mid x, a)$ to denote the element xz of matrix $\boldsymbol{O}_a$. A partially observable Markov decision process is fully specified by

- Its state space, $\mathcal{X}$;
- Its action space, $\mathcal{A}$;
- Its observation space, $\mathcal{Z}$;
- The transition probabilities, $\{\boldsymbol{P}_a, a \in \mathcal{A}\}$;
- The observation probabilities, $\{\boldsymbol{O}_a, a \in \mathcal{A}\}$;
- The immediate cost function, c;
- The initial state, x_0, or initial distribution over states, μ_0.[1]

Therefore, we can represent a partially observable Markov decision process as a tuple $(\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c, \mu_0)$ or, when the initial distribution is implicit, as a more compact tuple $(\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c)$.

It should be clear that any hidden Markov model is a particular case of a partially observable Markov decision process in which there is no actuation—or, equivalently, in which there is a single action and $c \equiv 0$. Similarly, any Markov decision processes is a particular case of a partially observable Markov decision process in which $\mathcal{Z} = \mathcal{X}$ and $\boldsymbol{O}_a = \boldsymbol{I}$. Figure 6.6 illustrates the relation between the different models surveyed so far.

Finally, a partially observable Markov decision process $(\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c)$ implicitly defines a Markov decision process $(\mathcal{X}, \mathcal{A}, \{\boldsymbol{P}_a\}, c)$, since neither the transition probabilities nor the immediate cost depend on the observation process

[1] We implicitly assume that the first observation, z_0, is subsumed in either the initial state x_0 or the initial distribution, μ_0.

$\{z_t, t \in \mathbb{N}\}$. We refer to $(\mathcal{X}, \mathcal{A}, \{\boldsymbol{P}_a\}, c)$ as the *underlying Markov decision process.*

6.2.1 Policies and optimality

As seen in Chapter 5, a *policy* is a mapping $\pi : \mathcal{H} \to \Delta(\mathcal{A})$ that maps each history $h \in \mathcal{A}$ to a distribution $\pi(h)$ over $\mathcal{A}$. As before, we write $\pi(a \mid h)$ to denote the probability of action a under policy π when history h has been observed.

Policy categorization remains essentially unchanged from Chapter 5, with a minor exception. In a partially observable Markov decision process histories are sequences of observations and actions, and include no (explicit) information on the hidden state. In other words, each $h \in \mathcal{H}$ is of the form

$$h = \{z_0, a_0, z_1, \dots, z_{t-1}, a_{t-1}, z_t\},$$

for some finite $t \in \mathbb{N}$. As such, in partially observable Markov decision problems, we say that a policy is *memoryless* if the probability $\pi(a \mid h)$ depends only on $|h|$ and on the last *observation* in h. We write $\pi_t(a \mid z)$ to denote the probability $\pi(a \mid h)$ when $|h| = t$ and the last observation in h is z. A memoryless policy thus ignores all past history of observations and actions. Clearly, memoryless policies and Markov policies coincide in (fully observable) Markov decision processes. A policy is stationary if it is memoryless and independent of $|h|$.

One key difference between a memoryless policy in a partially observable Markov decision process and a Markov policy in a (fully observable) Markov decision process is that the former does not induce a Markov chain, while the latter does. To see why, let $h = \{z_0, a_0, \dots, z_{t-1}, a_{t-1}, z_t\}$. Then,

$$\mathbb{P}[\mathrm{x}_{t+1} = y \mid \mathrm{h}_t = h] = \sum_{x,a} \mathbb{P}[\mathrm{x}_{t+1} = y \mid \mathrm{a}_t = a, \mathrm{x}_t = x, \mathrm{h}_t = h]$$
$$\mathbb{P}[\mathrm{a}_t = a \mid \mathrm{x}_t = x, \mathrm{h}_t = h]\, \mathbb{P}[\mathrm{x}_t = x \mid \mathrm{h}_t = h].$$

Using the Markov property and assuming a memoryless policy, we finally have that

$$\mathbb{P}[\mathrm{x}_{t+1} = y \mid \mathrm{h}_t = h] = \sum_{x,a} \boldsymbol{P}(y \mid x, a)\pi_t(a \mid z_t)\mathbb{P}[\mathrm{x}_t = x \mid \mathrm{h}_t = h],$$

showing that the probability governing x_{t+1} maintains its dependency on the history.

Optimality

Given a partially observable Markov decision process $\big\{\big((\mathrm{x}_t, \mathrm{z}_t), \mathrm{a}_t, \mathrm{c}_t\big), t \in \mathbb{N}\big\}$ the corresponding *expected discounted cost-to-go* (EDC) is the quantity

$$DC \stackrel{\text{def}}{=} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathrm{c}_t\right],$$

with $\gamma \in [0, 1]$. As in MDPs, the EDC provides a criterion to compare policies in a partially observable Markov decision processes. The decision problem of selecting a policy that minimizes the EDC in a partially observable Markov decision process is called a *partially observable Markov decision problem* (or POMDP). We represent a POMDP compactly as a tuple $(\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c, \gamma, \mu_0)$ or $(\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c, \gamma)$, if the initial distribution is implicit.

Given a POMDP $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c, \gamma)$ and a probability distribution μ over $\mathcal{X}$, the cost-to-go associated with a policy π and distribution μ is defined as

$$J^\pi(\mu) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t \mathrm{c}_t \mid \mathrm{x}_0 \sim \mu\right].$$

A policy π^* is *optimal* if

$$J^{\pi^*}(\mu) \leq J^\pi(\mu),$$

for any policy π and any distribution μ over $\mathcal{X}$. Clearly, if an optimal policy π^* exists, then

$$J^{\pi^*}(\mu) = \inf_{\pi \in \Pi^{\mathrm{MR}}} \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t \mathrm{c}_t \mid \mathrm{x}_0 \sim \mu\right],$$

for any distribution μ over $\mathcal{X}$.

6.2.2 Examples

We now go over a number of example problems that can be modeled using POMDPs. We postpone to Sections 6.4.3 and **??** the discussion of solutions for these problems. We start with the tiger problem, which is a standard problem in the POMDP literature.

The tiger problem

Consider the problem of a prisoner trying to escape a dungeon is faced with two similar looking doors. Behind one of the doors lies a tiger, and behind the other lies the path to freedom. Unfortunately, the agent does not know what fate lies behind each door.

Before selecting the door, the prisoner may listen behind the doors to figure out where tiger lies. However, listening is misleading and the delay may hasten the capture of the prisoner.

We model the prisoner's problem as a POMDP $(\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c, \gamma)$, where

- $\mathcal{X} = \{L, R\}$, where L and R correspond to the tiger on the left and on the right, respectively.
- $\mathcal{A} = \{\mathrm{OL}, \mathrm{OR}, \mathrm{L}\}$, corresponding to the actions "Open left door", "Open right door" and "Listen".
- Upon listening, the prisoner may hear the tiger on the left or on the right, yielding $\mathcal{Z} = \{L, R\}$.

- For modeling purposes, we consider that after opening either door, the tiger is replaced at random (perhaps for the next prisoner). The transition probabilities thus become

$$\boldsymbol{P}_{\text{OL}} = \boldsymbol{P}_{\text{OR}} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix} \qquad \text{and} \qquad \boldsymbol{P}_{\text{L}} = \begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix}.$$

- Replacing the tiger leads to an indistinct noise that may come from any of the two doors. We model this as a uniform probability of making any of the two observations. When listening, the prisioner will hear the tiger behind the correct door with a 0.85 probability. Formally,

$$\boldsymbol{O}_{\text{OL}} = \boldsymbol{O}_{\text{OR}} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix} \qquad \text{and} \qquad \boldsymbol{O}_{\text{L}} = \begin{bmatrix} 0.85 & 0.15 \\ 0.15 & 0.85 \end{bmatrix}.$$

- Finally, cost function can be represented as the matrix

$$\boldsymbol{C} = \begin{bmatrix} 1 & 0 & 0.01 \\ 0 & 1 & 0.01 \end{bmatrix}.$$

Web advertising

Consider the problem of designing an ad placement engine for a sports webpage. The system can place adds regarding two categories of sports products (water sports and mountain sports); it can select to (a) place an add for mountain sports; (b) place an ad for water sports; or (c) select a "neutral" ad (general sports products).

Showing the wrong category of products can cause the visitor to leave the webpage. Also, after buying a product, visitors always leave the site. Finally, to aid in the decision, the ad placement engine has the ability to track which products the visitor clicks to predict which type of customer (interested in water or mountain sports) the visitor is.

Modeling the decision process as a POMDP $(\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c, \gamma)$, we have that

- $\mathcal{X} = \{B, L, M, W\}$, where W and M correspond to a customer interested in water or mountain sports, respectively. B corresponds to the situation where the visitor buys the advertised product; L corresponds to the situation where the visitor leaves the website.
- $\mathcal{A} = \{M, N, W\}$, where W and M respectively correspond to placing a water or a mountain sports ad. N corresponds to the option of placing a neutral ad.
- $\mathcal{Z} = \{L, M, W\}$, where L corresponds to no visitor, and W and M correspond to interest (a click) in water sports or mountain sports, respectively.
- The transition probabilities can be summarized as

$$\boldsymbol{P}_M = \begin{bmatrix} 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.5 & 0.5 \\ 0.05 & 0.15 & 0.8 & 0.0 \\ 0.01 & 0.7 & 0.29 & 0.0 \end{bmatrix}, \quad \boldsymbol{P}_W = \begin{bmatrix} 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.5 & 0.5 \\ 0.01 & 0.7 & 0.0 & 0.29 \\ 0.10 & 0.15 & 0.0 & 0.75 \end{bmatrix},$$

and

$$\boldsymbol{P}_N = \begin{bmatrix} 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.5 & 0.5 \\ 0.01 & 0.4 & 0.59 & 0.0 \\ 0.01 & 0.4 & 0.0 & 0.59 \end{bmatrix}.$$

- The observation probabilities do not depend on the action selected by the agent, i.e., $\boldsymbol{O}_M = \boldsymbol{O}_N = \boldsymbol{O}_W = \boldsymbol{O}$, with

$$\boldsymbol{O} = \begin{bmatrix} 0.0 & 0.5 & 0.5 \\ 1.0 & 0.0 & 0.0 \\ 0.0 & 0.85 & 0.15 \\ 0.0 & 0.15 & 0.85 \end{bmatrix}.$$

  Note that the type of buying visitor is irrelevant and coveys no indication on the type of customer.

- Finally, the cost function can be represented as the matrix

$$\boldsymbol{C} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \end{bmatrix},$$

  where all alternatives (other than a buying customer) are costly.

Slotted ALOHA

ALOHAnet was one of the first wireless computer networking systems, developed in the early 1970s at the University of Hawaii (Abramson, 2009). The network used a communication protocol that is presently known as *pure ALOHA* and works by having all client computers in the network share a channel for transmission. A client computer sends its data without verifying the status of the channel. An acknowledgement/retransmission mechanism is used to deal with *collision*, i.e., situations in which a client computer tries to transmit data when the channel is already in use. Whenever a client computer detects a collision, it waits a random amount of time before retransmitting the packet that collided.

An improvement over the pure ALOHA protocol is known as *slotted ALOHA* (L. Roberts, 1975). In slotted ALOHA, time is segmented into time slots of fixed length, and packets can only be transmitted at the beginning of a time slot, in order to minimize collisions. For simplicity, we assume that each packet requires exactly one slot of time to be transmitted. Our goal is to design a system to schedule packet transmission in the network (that runs in each client computer).

Let us suppose that new packets arrive at a rate λ. Each client computer maintains the packets to be transmitted in a backlog and are only removed when transmission succeeds. For simplicity, we assume that the system as a whole has capacity to backlog a maximum of $N_{\max}$ packets.

At any time step t, let the r.v. n_t denote the total number of backlogged packets (in all client computers). Also, let the r.v. s_t denote the status of the channel, which can be *idle*, if no packets were transmitted in the previous time slot; *transmit*, if a packet was successfully transmitted in the previous time slot; and *collision*, if two or more packets collided in the previous time slot.

Suppose that each packet is transmitted with probability p. Then the probability of successfully transmitting a packet corresponds to the probability of a single packet being transmitted, and is given by

$$\rho(p,n) \overset{\text{def}}{=} \mathbb{P}_p\left[\textit{transmit} \mid \mathrm{n}_t = n\right] = \sum_{m=1}^{n} p(1-p)^{n-1} = pn(1-p)^{n-1}. \qquad (6.4)$$

Therefore, the value of p that maximizes the probability in (6.4) is $p = 1/\mathrm{n}_t$. Unfortunately, n_t is unknown to the client computers—they can only access the status of the channel.

We can represent the decision problem of the scheduler—selecting the transmission probability p at each time step t—as a POMDP $(\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c, \gamma)$. In particular,

- The state $\mathbf{x}_t$ is the pair $(\mathrm{n}_t, \mathrm{s}_t)$ that includes the number of packets currently backlogged in the system and the status of the channel. Therefore,

$$\mathcal{X} = \{0, \ldots, N_{\max}\} \times \{I, T, C\},$$

  where I, T and C represent the idle, the transmit and collision status of the channel.

- The action a_t at each step t is the probability of transmitting a packet. Ideally, such probability can take any real value between 0 and 1. However, for modeling purposes—and since the maximum number of backlogged packets is $N_{\max}$—we set

$$\mathcal{A} = \left\{\tfrac{1}{n}, n = 1, \ldots, N_{\max}\right\}.$$

  The transition probabilities associated with each action account both for the probability of success and the probability of arrival of new packets. Specifically, if $\boldsymbol{x} = (m, s)$ and $\boldsymbol{y} = (n, s')$, with $m, n \in \{0, \ldots, N_{\max}\}$ and $s, s' \in \{I, T, C\}$,

$$\begin{aligned}\boldsymbol{P}(\boldsymbol{y} \mid \boldsymbol{x}, a) &= \mathbb{P}\left[\mathrm{n}_{t+1} = m \mid \mathrm{n}_t = n, \mathrm{s}_{t+1} = s'\right]\mathbb{P}\left[\mathrm{s}_{t+1} = s' \mid \mathrm{n}_t = n, \mathrm{a}_t = a\right] \\ &= \begin{cases}\mathsf{Poisson}(m-n+1;\lambda)\rho(a,n) & \text{if } s' = T \\ \mathsf{Poisson}(m-n;\lambda)(1-\rho(a,n)) & \text{if } s' = C \\ \mathsf{Poisson}(m-n;\lambda)(1-a)^n & \text{if } s' = I\end{cases}.\end{aligned}$$

- The observation z_t corresponds to the status of the channel at time step t, i.e., $\mathrm{z}_t = S_t$, and is a deterministic function of the state.

- Finally, the system incurs a cost of $1/N_{\max}$ for each backlogged message.

Fault detection and repair

We again consider the problem of developing an automated fault detection mechanism for a complex machine. The machine is used to manufacture a sensible electronic component, and comprises a total of N distinct modules. The condition of the modules directly impacts the quality of the manufactured components, and the agent (the fault detection mechanism) must decide each day whether the machine should be allowed to work, producing the prescribed number of parts for that day, or should be stopped and disassembled for inspection or repair of some module. Ultimately, the goal of the agent is to maximize the number and quality of manufactured components while minimizing the possibility of the machine breaking down.

Each module can be in good, fair, and bad condition, or broken. For each day that the machine is allowed to work, each module deteriorates with a probability of λ_D. Conversely, repairing a module improves its condition with a probability μ_R. Once a module breaks, it can no longer be repaired; the machine cannot be used any more and must be replaced.

The agent can infer the state of the modules in the machine in two ways:

- When the machine is allowed to work, the agent can observe the quality of the produced components. If all modules in the machine function properly during the day, it will produce good quality components; conversely, if any of the modules malfunctions, it will produce bad quality components. The probability of a module malfunctioning increases as its state deteriorates.

- When the machine is stopped for inspection, the agent is able to observe whether each part is in good or bad condition, although this observation process sometimes fails.

Producing bad components has a cost of c_B per day. Inspecting requires disassembling the machine and costs c_I, with $c_I > c_B$. Repairing costs c_R, with $c_R > c_I$ and, finally, replacing the machine has the maximum cost, 1.

We model the maintenance problem as a POMDP $(\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c, \gamma)$. The state $\mathbf{x}_t$ corresponds to a N-tuple $(\mathrm{x}_{1,t}, \dots, \mathrm{x}_{N,t})$. Each $\mathrm{x}_{n,t}, n = 1, \dots, N$, denotes the condition of module n on day t and takes values in $\{0, 1, 2, 3\}$, where 0 corresponds a module in good condition, 1 to a module in fair condition, 2 to a module in bad condition and 3 to a broken module. Therefore, $\mathcal{X} = \{0, 1, 2, 3\}^N$.

A new decision epoch occurs each day, where the agent must decide which activity (manufacture, inspect, repair or replace) should be conducted that day. Therefore, there are a total of four actions, $\mathcal{A} = \{I, M, R_1, R_2\}$, where M denotes the "manufacture" action, I denotes the "inspect" action, R_1 for the "repair" action, and R_2 for the "replace" action. The different modules are independent, so the transition probabilities associated with each action $a \in \mathcal{A}$ can be factorized as

$$\boldsymbol{P}(\boldsymbol{y} \mid \boldsymbol{x}, a) = \prod_{n=1}^{N} P_n(y_n \mid x_n, a),$$

where

$$P_n(y_n \mid x_n, I) = \mathbb{I}\left[y_n = x_n\right],$$

i.e., the inspect action does not affect the condition of the parts. Moreover,

$$P_n(y_n \mid x_n, M) = \begin{cases} 1 - \lambda_D & \text{if } x_n \in \{0,1,2\} \text{ and } y_n = x_n \\ \lambda_D & \text{if } x_n \in \{0,1,2\} \text{ and } y_n = x_n + 1 \\ 1.0 & \text{if } x_n = y_n = 4 \\ 0.0 & \text{otherwise} \end{cases}$$

and

$$P_n(y_n \mid x_n, R_1) = \begin{cases} 1 - \mu_R & \text{if } x_n \in \{1,2\} \text{ and } y_n = x_n \\ \mu_R & \text{if } x_n \in \{1,2\} \text{ and } y_n = x_n - 1 \\ 1.0 & \text{if } x_n \in \{0,3\} \text{ and } y_n = x_n \\ 0.0 & \text{otherwise} \end{cases}.$$

Finally, replacing the machine implies that all modules will be in good condition, i.e.,

$$P_n(y_n \mid x_n, R_2) = \mathbb{I}\left[y_n = 0\right].$$

The observations made by the agent depend on the corresponding actions. For example, when the agent selects the action "inspect", it observes whether each module is in good or bad condition. We represent the observed condition of module n as a r.v. $\mathrm{z}_{n,t} \in \{0, 1\}$, where 0 and 1 represent good and bad condition upon inspection, respectively. Hence, the observation resulting from the inspect action consists of a tuple $(\mathrm{z}_{1,t}, \dots, \mathrm{z}_{N,t})$. In contrast, when the agent decides to manufacture, it observes only whether the produced components are of good or bad quality. We represent a good quality component as $\mathbf{0} \in \mathbb{R}^N$ and a bad quality component as $\mathbf{1} \in \mathbb{R}^N$. Finally, both repair and replace actions yield no observation, which we represent as $\mathbf{0} \in \mathbb{R}^N$. Thus, the observation $\mathbf{z}_t$ at time step t is a tuple $(\mathrm{z}_{1,t}, \dots, \mathrm{z}_{N,t})$, with each $\mathrm{z}_{n,t} \in \{0, 1\}$.

Each module n contributes independently to the observation $\mathbf{z}_t$. Therefore, the observation probabilities can be factorized as

$$\boldsymbol{O}(\boldsymbol{z} \mid \boldsymbol{x}, a) = \prod_{n=1}^{N} O_n(z_n \mid x_n, a),$$

where

$$O_n(z_n \mid x_n, R_1) = O_n(0 \mid x_n, R_2) = \mathbb{I}\left[z_n = 0\right],$$

since the repair and replace action yield no observation. Additionally, the probability of detecting a good module decreases with the condition of the module as

$$O_n(0 \mid x_n, I) = \begin{cases} 0.97 & \text{if } x_n = 0 \\ 0.80 & \text{if } x_n = 1 \\ 0.05 & \text{if } x_n = 2 \\ 0.02 & \text{if } x_n = 3 \end{cases},$$

with $O_n(1 \mid x_n, I) = 1 - O_n(0 \mid x_n, I)$. Finally, the probability of a module functioning properly (and thus contributing to produce a good quality component) decreases with the condition of the module, yielding

$$O_n(0 \mid x_n, M) = \begin{cases} 1.0 & \text{if } x_n = 0 \\ 0.95 & \text{if } x_n = 1 \\ 0.75 & \text{if } x_n = 2 \\ 0 & \text{if } x_n = 3 \end{cases}.$$

The cost is given by

$$c(\boldsymbol{x}, a) = \begin{cases} c_B p_B(\boldsymbol{x}) & \text{if } a = M \\ c_I & \text{if } a = I \\ c_R & \text{if } a = R_1 \\ 1 & \text{if } a = R_2 \end{cases},$$

where p_F is the probability of producing a bad quality component, which is given by

$$p_F(\boldsymbol{x}) = \prod_{n=1}^{N} O_n(1 \mid x_n, M).$$

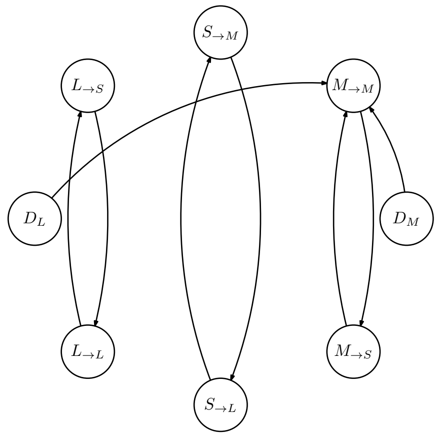
Space shuttle

Consider the situation of a shuttle delivering supplies to two space stations. The two stations are separated by a small distance; each station has a loading dock, to which the shuttle attaches to drop the supplies. We refer to the most recently visited station as the MRV station; the least recently visited station is referred as the LRV station. The shuttle always faces exactly one of the stations, and the autopilot system has 3 maneuvers available: Move forward, back up and turn around.

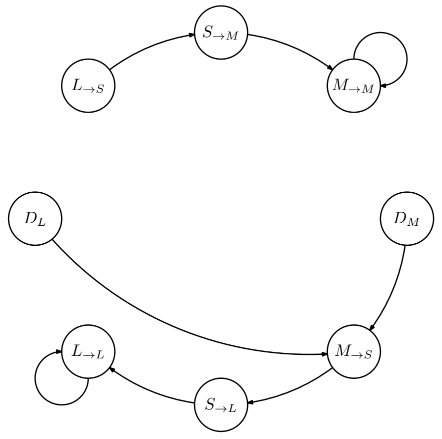
Moving forward from the docking station launches the shuttle into open space. If in open space, moving forward brings the shuttle close to the station that the shuttle is currently facing. Finally, if close to the station, moving forward drives the shuttle into the station, causing damage.

Baking up works inversely to moving forward. However, it is a more difficult maneuver and, as such, is often unsuccessful. Backing up from the docking station launches the shuttle to open space with a probability of success of only 0.3. If in open space, it approaches the station not faced by the shuttle with a probability 0.8. If next to the station, backing up attaches the shuttle to the station with a probability 0.7. When a maneuver fails, the position of the shuttle remains unchanged half of the times, but sometimes the shuttle turns and faces the opposite station. Finally, turning around makes the shuttle face the station that is not currently facing.

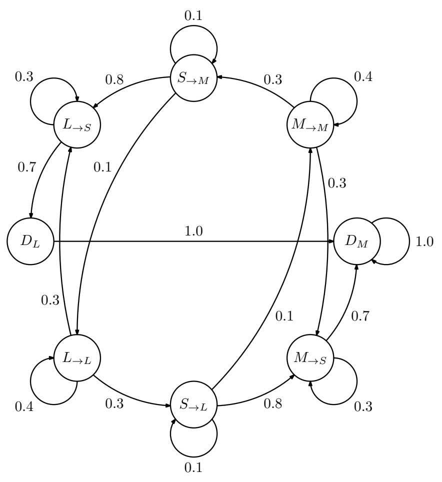
A successful docking motion has no cost, while a collision against the station costs 1. Every other motion (including unsuccessful ones) costs 0.75.



(a) Turn around motion. All transitions have probability 1.0.



(b) Move forward. All transitions have probability 1.0.



(c) Backup.

Figure 6.7 Transition probabilities for the different actions available to the autopilot system. Unlabeled transitions correspond to a probability of 1.0.

The sensors of the shuttle provide the autopilot system with very limited perception. When next to a station, it either perceives the station or empty space, depending where the shuttle is facing. In deep space, the station ahead is perceived only with a probability 0.7. When docked, only the interior of the dock is visible.

The shuttle scenario can be described as a partially observable Markov decision process $(\mathcal{X}, \mathcal{A}, \mathcal{Z}, \boldsymbol{P}, \boldsymbol{O}, r)$, where the state x_t corresponds to the position of the shuttle at each time step. The r.v. x_t can take values in the set $\mathcal{X} = \{D_L, M_{\to M}, S_{\to M}, L_{\to S}, M_{\to S}, S_{\to_L}, L_{\to L}, D_M\}$, where

- D_M and D_L correspond to the docked position, in the MRV and LRV, respectively.

- $A_{\to B}$ indicate that the shuttle is in position A facing B, with M denoting the MRV, L denoting the LRV and S denoting "space".

The actions available to the autopilot system correspond to the three maneuvers, "Turn around" (T), "Move forward" (F) and "Backup" (B), and the corresponding transitions are summarized in Fig. 6.7.

As for the observations, they depend only on the location of the shuttle, and correspond to what can be perceived by the sensors. Hence, at each time step t, the observation z_t takes values in the set $\mathcal{Z} = \{L, M, D_L, D_M, S\}$, where L and M indicate that LRV and MRV are visible from the shuttle, respectively; D_L and D_M indicate that the shuttle is docked in LRV and MRV, respectively; and S indicates that only space is visible from the shuttle. The observation probabilities can thus be computed as

$$\boldsymbol{O}(z \mid x, a) = \begin{cases} \mathbb{I}\left[x = L_{\to L}\right] + 0.7 \times \mathbb{I}\left[x = S_{\to L}\right] & \text{if } z = L \\ \mathbb{I}\left[x = M_{\to M}\right] + 0.7 \times \mathbb{I}\left[x = S_{\to M}\right] & \text{if } z = M \\ \mathbb{I}\left[x \in \{M_{\to S}, L_{\to S}\}\right] + 0.3 \times \mathbb{I}\left[x \in \{S_{\to L}, S_{\to M}\}\right] & \text{if } z = S \\ \mathbb{I}\left[x = D_L\right] & \text{if } z = D_L \\ t\mathbb{I}\left[x = D_M\right] & \text{if } z = D_M \end{cases}$$

Finally, the cost function follows trivially from the description,

$$c(x, a) = \begin{cases} 0 & \text{if } x = M \\ 1.0 & \text{if } x \in \{M_{\to M}, L_{\to L}\} \text{ and } a = F \\ 0.75 & \text{otherwise} \end{cases}$$

6.3 Belief MDP

As illustrated in Example 6.2, the optimal policy in a POMDP cannot ignore the hidden state. However, since the state is not directly observable, it must be estimated from the history of observations and actions. In other words, given the history up to time step t, h_t, we want to compute the probability

$$\mu_t(x) \overset{\text{def}}{=} \mathbb{P}\left[\mathrm{x}_t = x \mid \mathrm{h}_t = h, \mathrm{x}_0 \sim \mu_0\right], \tag{6.5}$$

for all $x \in \mathcal{X}$. The probability in (6.5) is similar to that computed in the HMM filtering problem (see Equation 3.3 in page 58). In fact, we can repeat the derivation in Chapter 3 to compute the distribution b_t. Let

$$\eta_t(x) \overset{\text{def}}{=} \mathbb{P}_{\mu_0}\left[\mathrm{x}_t = x, \mathrm{h}_t = h\right],$$

where the subscript μ_0 is shorthand notation to indicate that the probability above is conditioned on the fact that $\mathrm{x}_0 \sim \mu_0$, where μ_0 is the initial distribution for the

POMDP. Clearly, η_t plays a role similar to that of the forward mapping α_t in the theory of HMMs, and we have that

$$\mu_t(x) = \frac{\eta_t(x)}{\sum_{x' \in \mathcal{X}} \eta_t(x')}.$$

Moreover,, if $h = \{h', z, a\}$,

$$\begin{aligned}\eta_t(x) &= \mathbb{P}_{\mu_0}\left[\mathrm{x}_t = x, \mathrm{z}_t = z, \mathrm{a}_{t-1} = a, \mathrm{h}_{t-1} = h'\right] \\ &= \mathbb{P}_{\mu_0}\left[\mathrm{z}_t = z \mid \mathrm{x}_t = x, \mathrm{a}_{t-1} = a, \mathrm{h}_{t-1} = h'\right] \mathbb{P}_{\mu_0}\left[\mathrm{x}_t = x \mid \mathrm{a}_{t-1} = a, \mathrm{h}_{t-1} = h'\right].\end{aligned}$$

Using (6.2) and the total probability law (see Appendix A),

$$\begin{aligned}\eta_t(x) = \boldsymbol{O}(z \mid x, a) \sum_{y \in \mathcal{X}} \mathbb{P}_{\mu_0}\left[\mathrm{x}_t = x \mid \mathrm{x}_{t-1} = y, \mathrm{a}_{t-1} = a, \mathrm{h}_{t-1} = h'\right] \\ \cdot \mathbb{P}_{\mu_0}\left[\mathrm{x}_{t-1} = y \mid \mathrm{a}_{t-1} = a, \mathrm{h}_{t-1} = h'\right].\end{aligned}$$

Finally, using (6.1) the fact that a_{t-1} is a function of h_{t-1},

$$\begin{aligned}\eta_t(x) &= \boldsymbol{O}(z \mid x, a) \sum_{y \in \mathcal{X}} \boldsymbol{P}(x \mid y, a) \mathbb{P}_{\mu_0}\left[\mathrm{x}_{t-1} = y \mid \mathrm{h}_{t-1} = h'\right] \\ &= \boldsymbol{O}(z \mid x, a) \sum_{y \in \mathcal{X}} \boldsymbol{P}(x \mid y, a) \mu_{t-1}(y).\end{aligned}$$

Putting all together, we finally get that

$$\mu_t(x) = \frac{\sum_{y \in \mathcal{X}} \boldsymbol{O}(z \mid x, a) \boldsymbol{P}(x \mid y, a) \mu_{t-1}(y)}{\sum_{x', y \in \mathcal{X}} \boldsymbol{O}(z \mid x', a) \boldsymbol{P}(x' \mid y, a) \mu_{t-1}(y)}. \tag{6.6}$$

The importance of (6.6) is two-fold. First, it can be seen as the POMDP counterpart to the forward computation introduced in Chapter 3 for HMMs.

Second, and most importantly, (6.6) shows that the dependence of μ_t on the history h_{t-1} is subsumed in μ_{t-1}: given μ_{t-1}, no additional information regarding h_{t-1} is necessary to compute μ_t. This means that the distribution μ_t is a *sufficient statistic for the history* h_t, a fact that has profound implications in approaching POMDPs. In particular, given a POMDP $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a, \{O_a\}, c, \gamma\}$, we use the relation in (6.6) to derive an equivalent model—albeit more familiar—to $\mathcal{M}$.

We conclude by noting that we can associate with each finite history $h \in \mathcal{H}$ a distribution μ_h that can be computed recursively using (6.6).

6.3.1 Optimality and the belief MDP

Define, for any $t \in \mathbb{N}$, the r.v. $\mathbf{b}_t \in \mathbb{R}^{|\mathcal{X}|}$ with xth component given by

$$\mathrm{b}_{x,t} = \mathbb{P}_{\mu_0}\left[\mathrm{x}_t = x \mid \mathrm{h}_t\right].$$

We refer to $\mathbf{b}_t$ as the *belief* at time step t.[2] We often write $\mathrm{b}_{x,t}$ as $\mathbf{b}_t(x)$ to emphasize the fact that $\mathbf{b}_t(x)$ is the probability of x according to the distribution $\mathbf{b}_t$.

[2]In the literature, the distribution $\mathbf{b}_t$ is sometimes referred as a *belief state* or *information state*. We adopt the simpler designation of "belief", since $\mathbf{b}_{x,t}$ encodes the degree of belief that the agent holds about whether $\mathrm{x}_t = x$ after observing the history h_t.

Additionally, define the operator $\boldsymbol{B} : \Delta(\mathcal{X}) \times \mathcal{Z} \times \mathcal{A} \to \Delta(\mathcal{X})$ as follows. Given a distribution $\boldsymbol{b}$ over $\mathcal{X}$ and any $z \in \mathcal{Z}$ and $a \in \mathcal{A}$, $\boldsymbol{B}(\boldsymbol{b}, z, a)$ is the distribution with xth component

$$[\boldsymbol{B}(\boldsymbol{b}, z, a)]_x = \frac{\sum_{y \in \mathcal{X}} \boldsymbol{O}(z \mid x, a)\boldsymbol{P}(x \mid y, a)\boldsymbol{b}(y)}{\sum_{x', y \in \mathcal{X}} \boldsymbol{O}(z \mid x', a)\boldsymbol{P}(x' \mid y, a)\boldsymbol{b}(y)}.$$

The operator $\boldsymbol{B}$ is called the *belief update operator*; given an observation $z \in \mathcal{Z}$ and an action $a \in \mathcal{A}$, $\boldsymbol{B}$ performs the update in (6.6) on an arbitrary distribution $\boldsymbol{b}$ over $\mathcal{X}$.

Now the sequence of beliefs, $\{\mathbf{b}_t, t \in \mathbb{N}\}$, defines a stochastic process with the following important property that follows from (6.6).

Proposition 6.1. *The stochastic process $\{\mathbf{b}_t, t \in \mathbb{N}\}$ is a controlled Markov chain, i.e,*

$$\mathbb{P}\left[\mathbf{b}_{t+1} = \boldsymbol{b}' \mid \mathrm{a}_t = a, \mathrm{h}_t = h\right] = \mathbb{P}\left[\mathbf{b}_{t+1} = \boldsymbol{b}' \mid \mathrm{a}_t = a, \mathbf{b}_t = \boldsymbol{b}_h\right],$$

where $\boldsymbol{b}_h$ denotes the belief corresponding to the history h.

Proof. See Section 6.8. □

In particular, the transition probabilities for the Markov chain in Proposition 6.1 can be written as

$$\begin{aligned}\hat{\boldsymbol{P}}(\boldsymbol{b}' \mid \boldsymbol{b}, a) &\overset{\text{def}}{=} \mathbb{P}\left[\mathbf{b}_{t+1} = \boldsymbol{b}' \mid \mathrm{a}_t = a, \mathbf{b}_t = \boldsymbol{b}\right] \\ &= \sum_{z \in \mathcal{Z}} \sum_{x, y \in \mathcal{X}} \boldsymbol{b}(x)\boldsymbol{P}_a(y \mid x)\boldsymbol{O}_a(z \mid y)\mathbb{I}\left[\boldsymbol{b}' = \boldsymbol{B}(\boldsymbol{b}, z, a)\right].\end{aligned} \tag{6.7}$$

Let us consider in detail the implications of Proposition 6.1. Given a POMDP $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c, \gamma)$, let $\mathcal{B}$ denote the space of distributions over $\mathcal{X}$—henceforth referred as the *belief space*. Define the function $\hat{c} : \mathcal{B} \times \mathcal{A} \to [0, 1]$ as

$$c(\boldsymbol{b}, a) = \sum_{x \in \mathcal{X}} \boldsymbol{b}(x)c(x, a). \tag{6.8}$$

As seen in Proposition 6.1, the process $\{\mathbf{b}_t, t \in \mathbb{N}\}$ is a controlled Markov chain with state space $\mathcal{B}$ and with transition probabilities given by (6.7). Then, using the transition probabilities $\{\hat{\boldsymbol{P}}_a, a \in \mathcal{A}\}$ and the function $\hat{c}$ in (6.8), we can define the so-called belief MDP.

Belief MDP

Given a POMDP $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c, \gamma)$, the *belief MDP* associated with $\mathcal{M}$ is the MDP $\hat{\mathcal{M}} = (\mathcal{B}, \mathcal{A}, \{\hat{\boldsymbol{P}}_a\}, \hat{c}, \gamma)$, where

- $\mathcal{B} = \Delta(\mathcal{X})$, i.e., the state space is the set of all probability distributions over $\mathcal{X}$.

- The action space is the same as that of $\mathcal{M}$.
- The transition probabilities, $\hat{\boldsymbol{P}}_a$, are defined in (6.7).
- The immediate cost, $\hat{c}$, is defined in (6.8).
- The discount is the same as that of $\mathcal{M}$.

The relevance of the belief MDP associated with a POMDP stems from the fact that the former is equivalent to the latter, in terms of decision making. This is a central result in this chapter and is established in the continuation.

POMDP and belief MDP equivalence ($\star$)

For ease of exposition, we introduce some temporary notation. If $\mathcal{M}$ is a POMDP $(\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c, \gamma, \boldsymbol{b}_0)$ and $\hat{\mathcal{M}}$ the associated belief MDP, we denote entities in $\hat{\mathcal{M}}$ with a hat $\hat{}$, while entities in $\mathcal{M}$ are written without the hat. For example, the history up to time step t, in $\mathcal{M}$, is the r.v.

$$\mathrm{h}_t = \{\mathrm{z}_0, \mathrm{a}_0, \ldots, \mathrm{a}_{t-1}, \mathrm{z}_t\}$$

taking values in the set $\mathcal{H}$, while the history up to time step t, in $\hat{\mathcal{M}}$, is the r.v.

$$\hat{\mathrm{h}}_t = \{\mathbf{b}_0, \mathrm{a}_0, \ldots, \mathrm{a}_{t-1}, \mathbf{b}_t\}$$

taking values in the set $\hat{\mathcal{H}}$. Clearly, to every history $h \in \mathcal{H}$ corresponds a single history $\hat{h} \in \hat{\mathcal{H}}$.[3] We write $\boldsymbol{b}_h$ to denote the belief resulting from the history $h \in \mathcal{X}$ and $\hat{h}_h \in \hat{\mathcal{H}}$ to denote the belief-MDP history corresponding to h. Similarly, we write J^π to denote the cost-to-go function associated with a policy π in $\mathcal{M}$; conversely, we write $\hat{J}^{\hat{\pi}}$ to denote the cost-to-go function associated with a policy $\hat{\pi}$ in $\hat{\mathcal{M}}$.

We start by noting that, given any policy $\pi : \mathcal{H} \to \Delta(\mathcal{A})$,

$$\begin{aligned}
&J^\pi(\boldsymbol{b}_0) \overset{\text{def}}{=} \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t \mathrm{c}_t \mid \mathrm{x}_0 \sim \boldsymbol{b}_0\right] \\
&= \sum_{t=0}^{\infty} \sum_{x,a,\boldsymbol{b}} \gamma^t c(x,a) \mathbb{P}_\pi\left[\mathrm{x}_t = x, \mathrm{a}_t = a, \mathbf{b}_t = \boldsymbol{b} \mid \mathrm{x}_0 \sim \boldsymbol{b}_0\right] \\
&= \sum_{t=0}^{\infty} \sum_{x,a,\boldsymbol{b}} \gamma^t c(x,a) \mathbb{P}_\pi\left[\mathrm{x}_t = x \mid \mathrm{a}_t = a, \mathbf{b}_t = \boldsymbol{b}, \mathrm{x}_0 = \boldsymbol{b}_0\right] \mathbb{P}_\pi\left[\mathrm{a}_t = a, \mathbf{b}_t = \boldsymbol{b} \mid \mathrm{x}_0 \sim \boldsymbol{b}_0\right] \\
&= \sum_{t=0}^{\infty} \sum_{x,a,\boldsymbol{b}} \gamma^t c(x,a) \boldsymbol{b}(x) \mathbb{P}_\pi\left[\mathrm{a}_t = a, \mathbf{b}_t = \boldsymbol{b} \mid \mathrm{x}_0 \sim \boldsymbol{b}_0\right].
\end{aligned}$$

[3]The converse, however, is not true.

Finally, replacing the definition of $\hat{c}$, we get

$$J^\pi(\boldsymbol{b}_0) = \sum_{t=0}^{\infty} \sum_{a,\boldsymbol{b}} \gamma^t \hat{c}(\boldsymbol{b}, a) \mathbb{P}_\pi \left[\mathrm{a}_t = a, \mathbf{b}_t = \boldsymbol{b} \mid \mathrm{x}_0 \sim \boldsymbol{b}_0 \right]. \tag{6.9}$$

Repeating the same derivation, given any policy $\hat{\pi} : \hat{\mathcal{H}} \to \Delta(\mathcal{A})$, we also have that

$$\hat{J}^{\hat{\pi}}(\boldsymbol{b}_0) = \sum_{t=0}^{\infty} \sum_{a,\boldsymbol{b}} \gamma^t \hat{c}(\boldsymbol{b}, a) \mathbb{P}_{\hat{\pi}} \left[\mathrm{a}_t = a, \mathbf{b}_t = \boldsymbol{b} \mid \mathrm{x}_0 \sim \boldsymbol{b}_0 \right]. \tag{6.10}$$

◇

Let $\pi : \mathcal{H} \to \Delta(\mathcal{A})$ denote an arbitrary POMDP policy for $\mathcal{M}$, and define the belief MDP policy $\hat{\pi} : \hat{\mathcal{H}} \to \Delta(\mathcal{A})$ as

$$\begin{aligned} &\hat{\pi}(a \mid \boldsymbol{b}_{0:t}, \boldsymbol{a}_{0:t-1}) \\ &\qquad = \sum_{\boldsymbol{z}_{0:t}} \pi(a \mid \boldsymbol{z}_{0:t}, \boldsymbol{a}_{0:t-1}) \mathbb{P}_\pi \left[\mathbf{z}_{0:t} = \boldsymbol{z}_{0:t} \mid \mathbf{b}_{0:t} = \boldsymbol{b}_{0:t}, \mathbf{a}_{0:t-1} = \boldsymbol{a}_{0:t-1} \right]. \end{aligned}$$

We have the following result.

Lemma 6.2. *For any $t \in \mathbb{N}$,*

$$\mathbb{P}_{\hat{\pi}} \left[\mathrm{a}_t = a, \mathbf{b}_t = \boldsymbol{b} \mid \mathrm{x}_0 \sim \boldsymbol{b}_0 \right] = \mathbb{P}_\pi \left[\mathrm{a}_t = a, \mathbf{b}_t = \boldsymbol{b} \mid \mathrm{x}_0 \sim \boldsymbol{b}_0 \right]. \tag{6.11}$$

Proof. See Section 6.8. □

By virtue of (6.9), (6.10) and Lemma 6.2, it follows that, for every policy π on $\mathcal{M}$, there is a policy $\hat{\pi}$ on $\hat{\mathcal{M}}$ such that $J^\pi(\boldsymbol{b}) = \hat{J}^{\hat{\pi}}(\boldsymbol{b})$.

On the other hand, in spite of the fact that the MDP $\hat{\mathcal{M}}$ features a continuous state space, the results from Chapter 5 on the existence of a deterministic and stationary optimal policy still hold (see the discussion in Section 5.7 and references therein). Therefore, there is an optimal policy for $\hat{\mathcal{M}}$ that is deterministic and stationary. However, for any deterministic stationary policy $\hat{\pi}$ in $\hat{\mathcal{M}}$, the policy π defined as

$$\pi(a \mid h) = \hat{\pi}(a \mid \boldsymbol{b}_h)$$

selects, for every history, the same actions as $\hat{\pi}$. This means we can convert the optimal policy for $\hat{\mathcal{M}}$, $\hat{\pi}^*$, into an equivalent policy for $\mathcal{M}$. Let π^+ denote such policy. Then, for any $\boldsymbol{b} \in \mathcal{B}$,

$$J^\pi(\boldsymbol{b}) = \hat{J}^{\hat{\pi}}(\boldsymbol{b}) \leq \hat{J}^{\hat{\pi}^*}(\boldsymbol{b}) = J^{\pi^+}(\boldsymbol{b}),$$

and the policy π^+ is optimal for $\mathcal{M}$. We established the following result.

Theorem 6.3. *Given a POMDP $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c, \gamma)$, then the solution to the belief MDP $(\mathcal{B}, \mathcal{A}, \{\hat{\boldsymbol{P}}_a\}, \hat{c}, \gamma)$ is also a solution to $\hat{\mathcal{M}}$.*

Theorem 6.3 implies not only the existence of optimal policies for POMDPs, but also suggests that MDP solution methods can be applied, with due modifications, to compute that optimal policy for POMDPs—by computing the optimal policy for the associated belief MDP. Interestingly, as will soon become apparent, the optimal cost-to-go function for $\hat{\mathcal{M}}$ (or, equivalently, for $\mathcal{M}$) has a particular structure that allows the methods from Chapter 5 to be easily adapted to the belief space, in spite of its continuous nature.

6.4 Exact POMDP solution methods

Let $(\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c, \gamma)$ denote a POMDP and $(\mathcal{B}, \mathcal{A}, \{\hat{\boldsymbol{P}}_a\}, \hat{c}, \gamma)$ the corresponding belief MDP. Given the equivalence between the two established in Section 6.3, our goal is to derive solution methods for POMDPs from the corresponding solution methods for MDPs.

Given a policy $\pi : \mathcal{B} \to \mathcal{A}$, the corresponding cost-to-go function verifies

$$J^\pi(\boldsymbol{b}) = \sum_{a \in \mathcal{A}} \pi(\boldsymbol{b}, a) \left[\hat{c}(\boldsymbol{b}, a) + \gamma \sum_{\boldsymbol{b}' \in \mathcal{B}} \hat{\boldsymbol{P}}(\boldsymbol{b}' \mid \boldsymbol{b}, a) J^\pi(\boldsymbol{b}') \right].$$

Similarly, the optimal cost-to-go function verifies

$$J^*(\boldsymbol{b}) = \min_{a \in \mathcal{A}} \left[\hat{c}(\boldsymbol{b}, a) + \gamma \sum_{\boldsymbol{b}' \in \mathcal{B}} \hat{\boldsymbol{P}}(\boldsymbol{b}' \mid \boldsymbol{b}, a) J^*(\boldsymbol{b}') \right].$$

We can replace the definitions of $\hat{c}$ and $\hat{\boldsymbol{P}}$ to get equivalent expressions featuring the original POMDP parameters. In particular, for the cost-to-go J^π, we get

$$\begin{aligned} &J^\pi(\boldsymbol{b}) \\ &= \sum_{a \in \mathcal{A}} \pi(\boldsymbol{b}, a) \sum_{x \in \mathcal{X}} \boldsymbol{b}(x) \left[c(x, a) + \gamma \sum_{z \in \mathcal{Z}} \sum_{y \in \mathcal{X}} \boldsymbol{P}(y \mid x, a) \boldsymbol{O}(z \mid y, a) J^\pi(\boldsymbol{B}(\boldsymbol{b}, z, a)) \right] \end{aligned} \tag{6.12}$$

and, for J^*,

$$J^*(\boldsymbol{b}) = \min_{a \in \mathcal{A}} \sum_{x \in \mathcal{X}} \boldsymbol{b}(x) \left[c(x, a) + \gamma \sum_{z \in \mathcal{Z}} \sum_{y \in \mathcal{X}} \boldsymbol{P}(y \mid x, a) \boldsymbol{O}(z \mid y, a) J^*(\boldsymbol{B}(\boldsymbol{b}, z, a)) \right]. \tag{6.13}$$

Adapting value and policy iteration to POMDPs will surely rely on recursions (6.12) and (6.13). However, in both cases the cost-to-go function is defined over the continuous set $\mathcal{B}$, which poses difficulties in terms of representation. One possibility to deal with such difficulties would be to use approximate methods such as those discussed in Section 5.5. Fortunately, as we now discuss, POMDPs possess additional structure that can be leveraged by solution methods.

6.4.1 Value iteration

We start by discussing value iteration for control.

Control

To make evident the structure of J^*, we naively follow Algorithm 5.2. Let $J^{(0)} \equiv 0$. Then, from (6.13), we can write

$$J^{(1)}(\boldsymbol{b}) = \min_{a \in \mathcal{A}} \boldsymbol{b} \cdot \boldsymbol{\alpha}_a^{(1)},$$

where $\boldsymbol{\alpha}_a$ is a $|\mathcal{X}|$-dimensional vector with xth component

$$\boldsymbol{\alpha}_{x,a}^{(1)} = c(x,a) + \gamma \sum_{z \in \mathcal{Z}} \sum_{y \in \mathcal{X}} \boldsymbol{P}(y \mid x,a) \boldsymbol{O}(z \mid y,a) J^{(0)}(\boldsymbol{B}(\boldsymbol{b},z,a)) = c(x,a).$$

Repeating this process for $J^{(2)}$, we get

$$J^{(2)}(\boldsymbol{b}) = \min_{a \in \mathcal{A}} \boldsymbol{b} \cdot \boldsymbol{\alpha}_a^{(2)}$$

where, now,

$$\begin{aligned}\boldsymbol{\alpha}_{x,a}^{(2)} &= c(x,a) + \gamma \sum_{z \in \mathcal{Z}} \sum_{y \in \mathcal{X}} \boldsymbol{P}(y \mid x,a) \boldsymbol{O}(z \mid y,a) J^{(1)}(\boldsymbol{B}(\boldsymbol{b},z,a)) \\ &= c(x,a) + \gamma \sum_{z \in \mathcal{Z}} \sum_{y \in \mathcal{X}} \boldsymbol{P}(y \mid x,a) \boldsymbol{O}(z \mid y,a) \min_{a' \in \mathcal{A}} \boldsymbol{B}(\boldsymbol{b},z,a) \cdot \boldsymbol{\alpha}_{a'}^{(1)}.\end{aligned}$$

We observe that, in both situations, $J^{(k)}$ can be written in the form

$$J^{(k)}(\boldsymbol{b}) = \min_{\boldsymbol{\alpha} \in \Gamma} \boldsymbol{b} \cdot \boldsymbol{\alpha},$$

for some set Γ of vectors. The following result generalizes this observation.

Proposition 6.4. *Let $J^{(k)}$ denote the cost-to-go function obtained after k iterations of Algorithm 5.2, where $J^{(0)} \equiv 0$. Then, $J^{(k)}$ is* piecewise linear and concave *(PWLC), i.e., can be written in the form*

$$J^{(k)}(\boldsymbol{b}) = \min_{\boldsymbol{\alpha} \in \Gamma^{(k)}} \boldsymbol{b} \cdot \boldsymbol{\alpha}, \tag{6.14}$$

for some finite set of vectors, $\Gamma^{(k)}$. Moreover, $\Gamma^{(k)}$ can be computed recursively as

$$\Gamma^{(k)} = \bigcup_{a \in \mathcal{A}} \Gamma_a^{(k)}, \tag{6.15}$$

where[4]

$$\Gamma_a^{(k)} = \bigoplus_{z\in\mathcal{Z}} \Gamma_{a,z}^{(k)} \tag{6.16}$$

and

$$\Gamma_{a,z}^{(k)} = \left\{ \frac{1}{|\mathcal{Z}|}\boldsymbol{C}_{:,a} + \gamma \boldsymbol{P}_a \operatorname{diag}(\boldsymbol{o}_{z,a})\,\boldsymbol{\alpha}^{(k-1)}, \boldsymbol{\alpha}^{(k-1)} \in \Gamma^{(k-1)} \right\}. \tag{6.17}$$

Proof. See Section 6.8. □

Proposition 6.4 establishes two key facts. First, each $J^{(k)}$ admits a *finite representation* as a collection of vectors, that we henceforth call *α-vectors*. Second, that the set of α-vectors at iteration k can be computed from the set of α-vectors at iteration $k-1$.

However, the direct implementation of (6.15), (6.16) and (6.17) leads to an exponential growth of the number of α-vectors since, from the definitions, $|\Gamma_{a,z}^{(k)}| = |\Gamma^{(k-1)}|$ and $|\Gamma_a^{(k)}| = |\Gamma_{a,z}^{(k)}|^{|\mathcal{Z}|} = |\Gamma^{(k-1)}|^{|\mathcal{Z}|}$, which yields $|\Gamma^{(k)}| = |\mathcal{A}|\,|\Gamma^{(k-1)}|^{|\mathcal{Z}|}$. Fortunately, not all vectors generated by the process just described are necessary to represent the cost-to-go function $J^{(k)}$.

Example 6.3 We revisit the tiger problem from Section 6.2.2, where the prisoner must select between two doors, behind one of which lies a tiger.

The belief space for the tiger domain consists of all probability distributions over the set $\mathcal{X} = \{L, R\}$. Therefore, a belief $\boldsymbol{b} \in \mathcal{B}$ is a pair $\boldsymbol{b} = (p, 1-p)$, where p represents the probability that the tiger is on the left. Therefore, the belief space is, in reality, 1-dimensional.

As discussed, at any belief $\boldsymbol{b} \in \mathcal{B}$,

$$\begin{aligned} J^{(k)}(\boldsymbol{b}) &= \min_{\boldsymbol{\alpha}\in\Gamma^{(k)}} \boldsymbol{b}\cdot\boldsymbol{\alpha} \\ &= \min_{\boldsymbol{\alpha}\in\Gamma^{(k)}} [p\alpha_1 + (1-p)\alpha_2] \\ &= \min_{\boldsymbol{\alpha}\in\Gamma^{(k)}} [p(\alpha_1 - \alpha_2) + \alpha_2], \end{aligned}$$

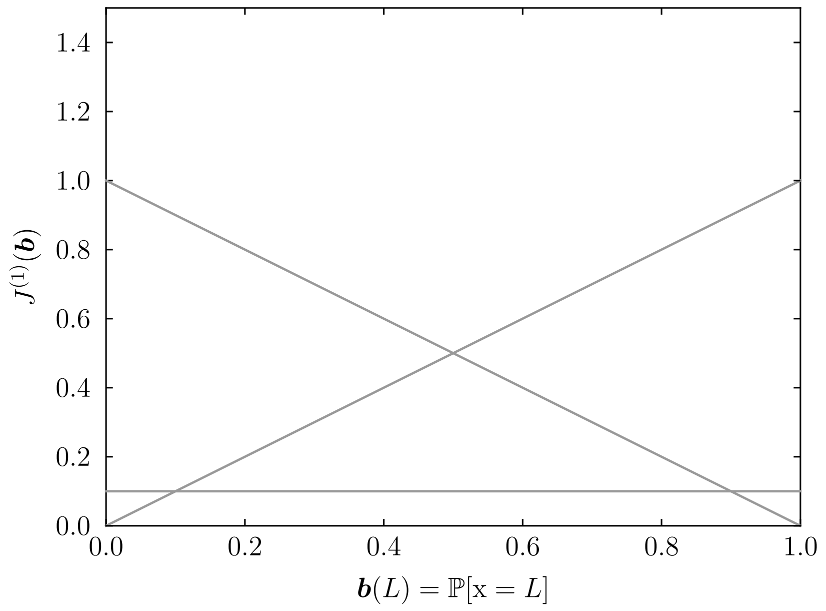
which corresponds to a line segment between α_2 (for $p = 0$) and α_1 (for $p = 1$). Therefore, we can plot the cost-to-go function $J^{(k)}$ as a function of p by plotting all segments $p(\alpha_1 - \alpha_2) + \alpha_2$ and, for each value of p, take the minimum.

Using (6.15), (6.16) and (6.17) to compute $\Gamma^{(1)}$ and $\Gamma^{(2)}$ yields the two functions depicted in Fig. 6.8. In each case, the function $J^{(k)}$ corresponds to the lower envelope across all vectors.

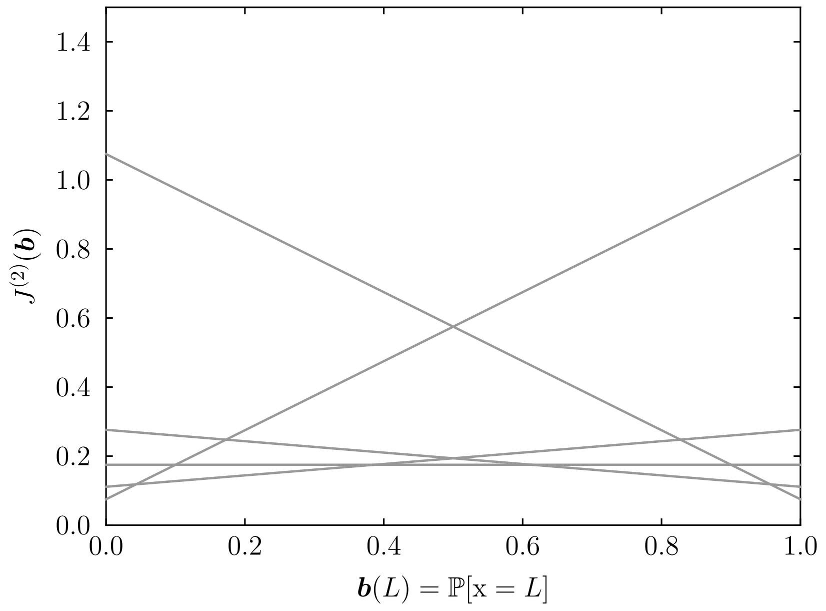
---

[4]Given two sets A and B, we write $A \oplus B$ to denote the *cross sum* of A and B, i.e., the set

$$A \oplus B = \{a + b \mid a \in A, b \in B\}.$$



(a) Cost-to-go estimate after one iteration. The α-vectors in the first iteration can be computed directly from cost matrix.



(b) Cost-to-go estimate after two iterations. The vectors depicted in black are sufficient to represent $J^{(2)}$. All other vectors (depicted in gray) are irrelevant.

Figure 6.8 Set of α-vectors computed during the computation of $J^{(1)}$ and $J^{(2)}$ for the tiger problem.

Figure 6.8 illustrates the fact that most of the α-vectors found by naively computing $\Gamma^{(k)}$ from $\Gamma^{(k-1)}$ are unnecessary to represent $J^{(k)}$. Therefore, exact implementations of VI for POMDPs maintain, at each iteration k, a *parsimonious representation of* $J^{(k)}$, in which useless vectors are ignored. We henceforth refer to the (parsimonious) representation of a PWLC function J to signify the minimal set of α-vectors needed to represent J.

Let $\Gamma^{(k-1)}$ denote the parsimonious representation of $J^{(k-1)}$. Let $\tilde{\Gamma}^{(k)}$ denote the set of all α-vectors computed from $\Gamma^{(k-1)}$ using (6.15), (6.16) and (6.17). We want to compute the minimal set $\Gamma^{(k)}$ that represents $\Gamma^{(k)}$. Clearly, we have that $\Gamma^{(k)} \subset \tilde{\Gamma}^{(k)}$. Moreover, a vector $\boldsymbol{\alpha}_0 \in \tilde{\Gamma}^{(k)}$ is an element of $\Gamma^{(k)}$ if and only if there is a non-empty region $\mathcal{R}(\boldsymbol{\alpha}_0) \subset \mathcal{B}$ where $\boldsymbol{\alpha}_0$ is strictly dominated by all other α-vectors, i.e., if

$$\boldsymbol{b} \cdot \boldsymbol{\alpha}_0 < \boldsymbol{b} \cdot \boldsymbol{\alpha},$$

for all $\boldsymbol{b} \in \mathcal{R}(\boldsymbol{\alpha}_0)$ and all $\boldsymbol{\alpha} \in \tilde{\Gamma}^{(k)} - \{\boldsymbol{\alpha}_0\}$. We call $\mathcal{R}(\boldsymbol{\alpha}_0)$ the *witness region for* $\boldsymbol{\alpha}_0$ and any belief $\boldsymbol{b} \in \mathcal{R}(\boldsymbol{\alpha}_0)$ as a *witness for* $\boldsymbol{\alpha}_0$.

The parsimonious representation of $J^{(k)}$ can be obtained by excluding from $\tilde{\Gamma}^{(k)}$ all α-vectors for which the witness region is empty. Exact VI methods build upon this observation and can broadly be grouped into two main categories:

Region-based methods This class of methods starts with an empty $\Gamma^{(k)}$ and adds only α-vectors for which the corresponding witness regions are non-empty. The most prominent method in this class is the *witness algorithm.*

Pruning methods This class of methods starts with $\Gamma^{(k)} = \tilde{\Gamma}^{(k)}$ and then eliminates (prunes) from $\Gamma^{(k+1)}$ all α-vectors with an empty witness region. The most prominent method in this class is the *incremental pruning algorithm.*

◇

Let $\Gamma^{(k-1)}$ denote the parsimonious representation of $J^{(k-1)}$, obtained after $k-1$ iterations of VI. For any given belief $\boldsymbol{b} \in \mathcal{B}$,

$$J^{(k)}(\boldsymbol{b}) = \min_{a \in \mathcal{A}} \sum_{x \in \mathcal{X}} \boldsymbol{b}(x) \left[c(x,a) + \gamma \sum_{z \in \mathcal{Z}} \sum_{y \in \mathcal{X}} \boldsymbol{P}(y \mid x,a) \boldsymbol{O}(z \mid y,a) J^{(k-1)}(\boldsymbol{B}(\boldsymbol{b},z,a)) \right].$$

Let $\boldsymbol{\alpha}^*(\boldsymbol{b},z,a)$ be such that

$$\boldsymbol{\alpha}^*(\boldsymbol{b},z,a) = \operatorname*{argmin}_{\boldsymbol{\alpha} \in \Gamma^{(k)}} \boldsymbol{B}(\boldsymbol{b},z,a) \cdot \boldsymbol{\alpha}.^5 \tag{6.18}$$

Moreover, let

$$\boldsymbol{\alpha}_{z,a}(\boldsymbol{b}) = \frac{1}{|\mathcal{Z}|} \boldsymbol{C}_{:,a} + \gamma \boldsymbol{P}_a \operatorname{diag}(\boldsymbol{o}_{z,a}) \boldsymbol{\alpha}^*(\boldsymbol{b},z,a)$$

[5]We assume the existence of some predefined order $\prec$ in $\mathcal{X}$. Such order, in turn, induces a lexicographic order in $\mathbb{R}^{|\mathcal{X}|}$ according to which $\boldsymbol{v}_1 \prec \boldsymbol{v}_2$ ($\boldsymbol{v}_1, \boldsymbol{v}_2 \in \mathbb{R}^{|\mathcal{X}|}$) if $v_{1,x} < v_{2,x}$ for some $x \in \mathcal{X}$ and $v_{1,x'} = v_{2,x'}$ for all $x' \prec x$. Therefore, if multiple α-vectors minimize (6.18), the (lexicographic) minimum is selected. That such a minimum always exists was proven by Littman (1996).

and

$$\boldsymbol{\alpha}_a(\boldsymbol{b}) = \sum_{z\in\mathcal{Z}} \boldsymbol{\alpha}_{z,a}(\boldsymbol{b}).$$

By construction, $\boldsymbol{\alpha}_{z,a}(\boldsymbol{b}) \in \Gamma^{(k)}_{z,a}$ and $\boldsymbol{\alpha}_a(\boldsymbol{b}) \in \Gamma^{(k)}_a$. We say that a vector $\boldsymbol{\alpha}'_a \in \Gamma^{(k)}_a$ is a *neighbor* of $\boldsymbol{\alpha}_a(\boldsymbol{b})$ if there is $z' \in \mathcal{Z}$ such that

$$\boldsymbol{\alpha}'_a = \sum_{z\neq z'} \boldsymbol{\alpha}_{z,a}(\boldsymbol{b}) + \boldsymbol{\alpha}',$$

where $\boldsymbol{\alpha}' \in \Gamma^{(k)}_{a,z'}$. We write $N(\boldsymbol{\alpha}_a(\boldsymbol{b}))$ to denote the set of all neighbors of $\boldsymbol{\alpha}_a(\boldsymbol{b})$. An interesting fact (see Lemma 6.11 in Section 6.8) is that, if there is a belief $\boldsymbol{b}' \neq \boldsymbol{b}$ such that $\boldsymbol{\alpha}^*(\boldsymbol{b}') \neq \boldsymbol{\alpha}^*(\boldsymbol{b})$, then there is a vector $\boldsymbol{\alpha} \in N(\boldsymbol{\alpha}^*(\boldsymbol{b}))$ such that

$$\boldsymbol{b}' \cdot \boldsymbol{\alpha} < \boldsymbol{b}' \cdot \boldsymbol{\alpha}^*(\boldsymbol{b}). \tag{6.19}$$

Also by construction, we have that

$$J^{(k)}(\boldsymbol{b}) = \min_{a\in\mathcal{A}} \boldsymbol{b} \cdot \boldsymbol{\alpha}_a(\boldsymbol{b}). \tag{6.20}$$

The α-vector that represents $J^{(k)}$ at $\boldsymbol{b}$ is the minimizer of (6.20),[6] and we denote such vector as $\boldsymbol{\alpha}^*(\boldsymbol{b})$. We are now in position to introduce the witness algorithm.

Algorithm 6.1 Witness algorithm to compute $\Gamma^{(k)}$ from $\Gamma^{(k-1)}$.

```
Require: POMDP model $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c, \gamma)$
Require: Parsimonious representation $\Gamma^{(k-1)}$
 1: Select arbitrary $\boldsymbol{b} \in \mathcal{B}$
 2: $\Gamma^{(k)} \leftarrow \{\boldsymbol{\alpha}^*(\boldsymbol{b})\}$
 3: $\tilde{\Gamma} \leftarrow N(\boldsymbol{\alpha}^*(\boldsymbol{b}))$
 4: while $\tilde{\Gamma} \neq \emptyset$ do
 5:     Select $\boldsymbol{\alpha} \in \tilde{\Gamma}$
 6:     if $\boldsymbol{\alpha} \in \Gamma^{(k)}$ then
 7:         $\boldsymbol{b} \leftarrow \perp$
 8:     else
 9:         $\boldsymbol{b} \leftarrow \mathsf{witness}(\boldsymbol{\alpha}, \Gamma^{(k)})$
10:     end if
11:     if $\boldsymbol{b} \neq \perp$ then
12:         $\Gamma^{(k)} \leftarrow \Gamma^{(k)} \cup \{\boldsymbol{\alpha}^*(\boldsymbol{b})\}$
13:         $\tilde{\Gamma} \leftarrow \tilde{\Gamma} \cup N(\boldsymbol{\alpha}^*(\boldsymbol{b}))$
14:     else
15:         $\tilde{\Gamma} \leftarrow \tilde{\Gamma} - \{\boldsymbol{\alpha}\}$
16:     end if
17: end while
18: return $\Gamma^{(k)}$
```

Let us consider Algorithm 6.1 in detail. The algorithm departs from an arbitrary belief $\boldsymbol{b}$ and adds the corresponding α-vector, $\boldsymbol{\alpha}^*(\boldsymbol{b})$, to $\Gamma^{(k)}$. The cycle

[6]Once again, in the presence of multiple minimizers, the lexicographic minimum is selected.

between lines 4 and 17 follows from (6.19): beliefs not in $\mathcal{R}(\boldsymbol{\alpha}^*(\boldsymbol{b}))$ can be found by searching witnesses for the neighbors of $\boldsymbol{\alpha}^*(\boldsymbol{b})$. Finally, the witness function in line 9 finds a witness for the vector $\boldsymbol{\alpha}$ given the set $\tilde{\Gamma} - \{\boldsymbol{\alpha}\}$, which amounts to solving the linear program

$$\begin{aligned} \text{maximize} \quad & \delta \\ \text{subject to} \quad & \boldsymbol{b} \cdot \boldsymbol{\alpha} + \delta \leq \boldsymbol{b} \cdot \boldsymbol{\alpha}', \quad \boldsymbol{\alpha}' \in \tilde{\Gamma} - \{\boldsymbol{\alpha}\} \\ & \boldsymbol{b} \cdot \mathbf{1} = 1 \\ & b_x \geq 0, \quad x \in \mathcal{X} \end{aligned} \tag{6.21}$$

where the optimization variables are $\boldsymbol{b}$ and δ. If there is a solution $(\boldsymbol{b}, \delta)$ for (6.21) in which $\delta > 0$, then $\boldsymbol{b}$ is a witness for $\boldsymbol{\alpha}$ and the function returns $\boldsymbol{b}$. Otherwise, the function returns $\perp$. The following result guarantees that the witness algorithm correctly builds the parsimonious representation for $J^{(k)}$ from $\Gamma^{(k-1)}$.

Proposition 6.5. *Given a POMDP $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c, \gamma)$ and a parsimonious representation $\Gamma^{(k-1)}$ for $J^{(k-1)}$, the witness algorithm correctly constructs the parsimonious representation $\Gamma^{(k)}$ for $J^{(k)}$.*

Proof. See Section 6.8. □

◇

In contrast with the witness algorithm, incremental pruning is a pruning method that proceeds by computing the set of vectors $\tilde{\Gamma}_a^{(k)}, a \in \mathcal{A}$, and then pruning the unnecessary vectors to obtain $\Gamma^{(k)}$. In particular, let prune denote the function that eliminates unnecessary α-vectors. We have that

$$\Gamma^{(k)} = \mathsf{prune}\left(\bigcup_{a \in \mathcal{A}} \Gamma_a^{(k)}\right)$$

and

$$\Gamma_a^{(k)} = \mathsf{prune}\left(\bigoplus_{z \in \mathcal{Z}} \tilde{\Gamma}_{a,z}^{(k)}\right).$$

A useful property of the cross-sum operator is that

$$\mathsf{prune}\left(\bigoplus_{z \in \mathcal{Z}} \tilde{\Gamma}_{a,z}^{(k)}\right) = \mathsf{prune}\left(\bigoplus_{z \in \mathcal{Z}} \mathsf{prune}(\tilde{\Gamma}_{a,z}^{(k)})\right).$$

Therefore, we can write

$$\Gamma_a^{(k)} = \mathsf{prune}\Big(\ldots \mathsf{prune}\big(\mathsf{prune}(\tilde{\Gamma}_{a,z_1}^{(k)} \oplus \tilde{\Gamma}_{a,z_2}^{(k)}) \oplus \tilde{\Gamma}_{a,z_3}^{(k)}\big) \ldots \oplus \tilde{\Gamma}_{a,z_{|\mathcal{Z}|}}^{(k)}\Big)$$

In this sense, pruning is performed incrementally, and the resulting algorithm is summarized as Algorithm 6.2.

Algorithm 6.2 Incremental pruning algorithm to compute $\Gamma^{(k)}$ from $\Gamma^{(k-1)}$.

```
Require: POMDP model $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c, \gamma)$
Require: Parsimonious representation $\Gamma^{(k-1)}$
 1: for $a \in \mathcal{A}$ do
 2:     $\Gamma_a^{(k)} \leftarrow \emptyset$
 3:     for $z \in \mathcal{Z}$ do
 4:         Compute $\tilde{\Gamma}_{z,a}^{(k)}$ using (6.17)
 5:         $\Gamma_a^{(k)} \leftarrow \mathsf{prune}(\Gamma_a^{(k)} \oplus \tilde{\Gamma}_{z,a}^{(k)})$
 6:     end for
 7: end for
 8: $\Gamma^{(k)} \leftarrow \mathsf{prune}\left(\bigcup_{a \in \mathcal{A}} \Gamma_a^{(k)}\right)$
 9: return $\Gamma^{(k)}$

10: function $\mathsf{prune}(\tilde{\Gamma})$
11:     $\Gamma \leftarrow \emptyset$
12:     while $\tilde{\Gamma} \neq \emptyset$ do
13:         Select $\boldsymbol{\alpha} \in \tilde{\Gamma}$
14:         $\boldsymbol{b} \leftarrow \mathsf{witness}(\boldsymbol{\alpha}, \Gamma)$
15:         if $\boldsymbol{b} \neq \perp$ then
16:             $\Gamma \leftarrow \Gamma \cup \{\boldsymbol{\alpha}^*(\boldsymbol{b})\}$
17:             $\tilde{\Gamma} \leftarrow \tilde{\Gamma} - \{\boldsymbol{\alpha}^*(\boldsymbol{b})\}$
18:         else
19:             $\tilde{\Gamma} \leftarrow \tilde{\Gamma} - \{\boldsymbol{\alpha}\}$
20:         end if
21:     end while
22:     return $\Gamma^{(k)}$
23: end function
```

Note that the guarantees in Proposition 6.5 hold true for incremental pruning *by construction*, i.e., Algorithm 6.2 correctly constructs the parsimonious representation $\Gamma^{(k)}$ for $J^{(k)}$ from $\Gamma^{(k-1)}$.

We conclude with a few important observations. First, both witness and incremental pruning algorithms described in Algorithms 6.1 and 6.2 correspond to *one update* of value iteration, producing a representation for $J^{(k)}$ from a representation of $J^{(k-1)}$.

Second, both witness and incremental pruning algorithms rely on the computation of witnesses to either (i) include a vector in Γ; or (ii) remove a vector from $\tilde{\Gamma}$. Computing a witness consists in solving the linear program in (6.21), which has a number of constraints that grows with $|\Gamma^{(k-1)}|$. Therefore, it comes as no surprise that both approaches quickly become prohibitive in terms of computational effort, and cannot be used but for the smallest problems.

Finally, each iteration of VI produces an estimate of J^* using a finite PWLC representation. However, in many practical problems, J^* does not admit a finite representation, requiring an infinite number of α-vectors to be represented exactly. Nevertheless, for sufficiently large k, $J^{(k)}$ provides a "sufficiently good"

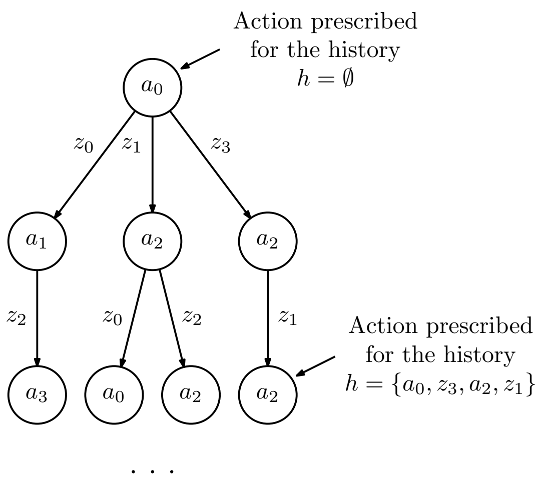

Action prescribed for the history $h = \emptyset$
a_0
z_0
z_1
z_3
a_1
a_2
a_2
z_2
z_0
z_2
z_1
Action prescribed for the history $h = \{a_0, z_3, a_2, z_1\}$
a_3
a_0
a_2
a_2
. . .

Figure 6.9 Example of a possible policy tree. The actions at each node are those prescribed by the policy represented by the tree for the history starting at the root and finishing in that node.

approximation of J^* in the sense that, for any $\varepsilon > 0$, there is a finite K such that $\|J^* - J^{(K)}\| \leq \varepsilon$. Such K can be computed as discussed in Chapter 5. Additionally, there is a restricted set of POMDPs for which J^* admits a finite representation of the form (6.14). Such POMDPs are called *finitely transient* and will be further discussed in Section 6.4.2.

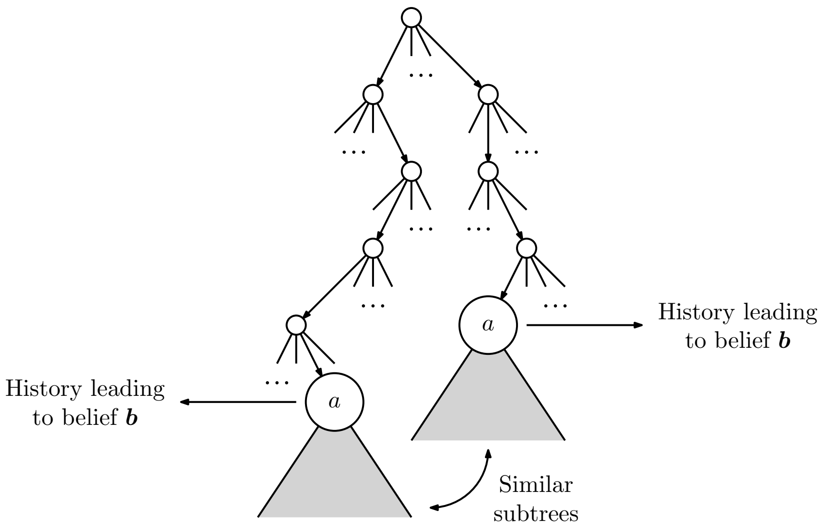
Prediction

We now consider the problem of *prediction*, i.e., computing the cost-to-go associated with a given POMDP policy. In light of our results from Section 6.3, we restrict our discussion to deterministic policies.

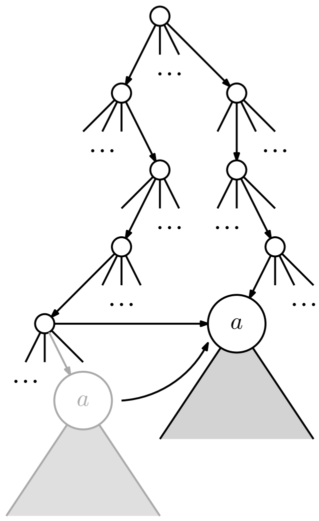
Given the equivalence between POMDPs and their associated belief MDPs established in Section 6.3, we can think of a policy either as a mapping $\pi : \mathcal{H} \to \mathcal{A}$ from histories to actions or, equivalently, as mapping $\pi : \mathcal{B} \to \mathcal{A}$ from beliefs to actions. Notwithstanding, the former interpretation of a policy—as a mapping $\pi : \mathcal{H} \to \mathcal{A}$—is more amenable to manipulation.

A policy $\pi : \mathcal{H} \to \mathcal{A}$ can be represented as a tree. Nodes in the tree are labeled with actions in $\mathcal{A}$, while branches are labeled with observations in $\mathcal{Z}$. The path from the root to a node a thus corresponds to a sequence of observations and actions—a history—for which the policy prescribes action a. Figure 6.9 illustrates the concept of policy tree.

A tree representation, however, generally includes redundant information. In fact, any two histories resulting in the same belief are equivalent. In terms of the policy tree, any two such histories correspond to distinct nodes that, however, have



(a) Original policy tree.



(b) The two equivalent nodes are “merged” into a single node.

Figure 6.10 Policy tree in which the two larger nodes are equivalent (they yield the same belief). In terms of policy representation, the two nodes can, therefore, be merged into a single node. The resulting structure is no longer a tree.

the same action label and similar subtrees. Therefore, the two nodes are redundant and can be collapsed into a single node, as depicted in Fig. 6.10. The resulting structure is no longer a tree but a *graph*, which is the most common representation for POMDP policies.

Let then π denote an arbitrary policy represented as a finite graph with a total of M nodes. We write $\text{act}(m)$ to denote the action label of node m and $\text{succ}(m, z)$ to denote the successor node of m when observation z is made. Each node m in the policy graph represents some region $\mathcal{R}_m$ in the belief-space $\mathcal{B}$ in which the prescribed action is $\text{act}(m)$; upon executing $\text{act}(m)$, the agent will make some observation $z \in \mathcal{Z}$ and move to node $\ell = \text{succ}(m, z)$, corresponding to region $\mathcal{R}_\ell$. It will then select action $\text{act}(\ell)$, and the process repeats.

It turns out that, for a policy represented as a finite graph, the cost-to-go function J^π takes the form

$$J^\pi(\boldsymbol{b}) = \boldsymbol{b} \cdot \boldsymbol{\alpha}(\boldsymbol{b}),$$

where $\boldsymbol{\alpha}(\boldsymbol{b}) \in \Gamma$ and Γ is a finite set of α-vectors, one for each node in the policy graph. The α-vector $\boldsymbol{\alpha}(\boldsymbol{b})$ denotes the one defined over the region containing $\boldsymbol{b}$. We thus recover a piecewise linear representation for J^π in which the corresponding α-vectors can be computed by solving a linear system of equations

$$\alpha_{m,x} = c(x, \text{act}(m)) + \gamma \sum_{z \in \mathcal{Z}} \sum_{y \in \mathcal{X}} \boldsymbol{P}(y \mid x, \text{act}(m)) \boldsymbol{O}(z \mid y, \text{act}(m)) \alpha_{\text{succ}(m,z),y}, \tag{6.22}$$

for each $x \in \mathcal{X}$ and $m \in \{1, \ldots, M\}$.

To show that the α-vectors that solve (6.22) indeed provide the desired representation for J^π, suppose that $\boldsymbol{\alpha}(\boldsymbol{b}) = \boldsymbol{\alpha}_m$, for some $m \in \{1, \ldots, M\}$. In other words, all histories $h \in \mathcal{H}$ that result in belief $\boldsymbol{b}$ correspond to the node m in the graph representation of π. Suppose then that $z \in \mathcal{Z}$ is observed after performing action $\text{act}(m)$. Then, it must hold that $\boldsymbol{B}(\boldsymbol{b}, z, \text{act}(m)) \in \mathcal{R}_{\text{succ}(m,z)}$. Multiplying both sides of (6.22) by $\boldsymbol{b}$ yields

$$\begin{aligned} \boldsymbol{b} \cdot \boldsymbol{\alpha}_m &= \sum_{x \in \mathcal{X}} \boldsymbol{b}(x) \Bigg[c(x, \text{act}(m)) + \gamma \sum_{y \in \mathcal{X}} \sum_{z \in \mathcal{Z}} \boldsymbol{P}(y \mid x, \text{act}(m)) \\ &\qquad\qquad \boldsymbol{O}(z \mid y, \text{act}(m)) \boldsymbol{\alpha}_{\text{succ}(m,z),y} \Bigg] \\ &= \sum_{x \in \mathcal{X}} \boldsymbol{b}(x) \Bigg[c(x, \text{act}(m)) + \gamma \sum_{y \in \mathcal{X}} \sum_{z \in \mathcal{Z}} \boldsymbol{P}(y \mid x, \text{act}(m)) \\ &\qquad\qquad \boldsymbol{O}(z \mid y, \text{act}(m)) [\boldsymbol{B}(\boldsymbol{b}, z, \text{act}(m)) \cdot \boldsymbol{\alpha}(\boldsymbol{B}(\boldsymbol{b}, z, \text{act}(m)))]_y \Bigg], \end{aligned}$$

which, by replacing $\boldsymbol{b} \cdot \boldsymbol{\alpha}(\boldsymbol{b})$ by $J(\boldsymbol{b})$ and $\boldsymbol{B}(\boldsymbol{b}, z, \text{act}(m)) \cdot \boldsymbol{\alpha}(\boldsymbol{B}(\boldsymbol{b}, z, \text{act}(m)))$ by $J(\boldsymbol{B}(\boldsymbol{b}, z, \text{act}(m)))$ yields precisely (6.12). Since J^π is the only solution for the recursion in (6.12), we can conclude that—as anticipated—the α-vectors that solve (6.22) provide the desired representation for J^π.

We conclude by noting that it is also possible to use an iterative approach to solve the system (6.22), akin to Algorithm 5.1 of Chapter 5. Such approach, however, is seldom used in practice.

6.4.2 Policy iteration

The policy iteration algorithm discussed in Chapter 5 (Algorithm 5.4) repeatedly goes through two steps:

(i) A *policy evaluation step* in which, given a policy $\pi^{(k)}$, the associated cost-to-go function, $J^{(k)}$, is computed.

(ii) A *policy improvement step* in which, given $J^{(k)}$, an improved policy $\pi^{(k+1)}$ is computed as

$$\pi^{(k+1)} = \pi_g^{J^{(k)}}.$$

The algorithm can be applied directly to belief MDPs, with due modifications that account for the continuous nature of the belief space. In particular, we discussed in Section 6.4.1 how to compute the cost-to-go function associated with a policy represented as a finite graph. That exact same approach can be used in step (i). As for step (ii), in order to compute an improved policy we leverage the close relation between policy graphs and piecewise linear function representations and replicate the steps used to compute $J^{(k+1)}$ from $J^{(k)}$ in VI.

Given some policy π represented as finite graph $\mathcal{G}$, let Γ denote the set of α-vectors used to represent J^π. We can apply any of the algorithms discussed in Section 6.4.1 (for example, incremental pruning) to compute an improved set of α-vectors from Γ, hereby denoted as Γ_{aux}. Each vector $\boldsymbol{\alpha}_m \in \Gamma_{\text{aux}}$ takes the form

$$\boldsymbol{\alpha}_m = \boldsymbol{C}_{:,a} + \gamma \sum_{\ell=1}^{|\mathcal{Z}|} \boldsymbol{P}_a \operatorname{diag}(\boldsymbol{o}_{z_\ell,a})\boldsymbol{\alpha}_\ell, \tag{6.23}$$

for some $a \in \mathcal{A}$ and some set of vectors $\{\boldsymbol{\alpha}_\ell, \ell = 1, \dots, |\mathcal{Z}|\} \subset \Gamma^{(k)}$. Therefore, for each $\boldsymbol{\alpha}_m \in \Gamma_{\text{aux}}$ we add a node m to $\mathcal{G}$, where $\text{act}(m) = a$ and $\text{succ}(m, z_\ell) = \ell$. The resulting graph is then "cleaned up" by removing unnecessary nodes:

- If a vector $\boldsymbol{\alpha}_m \in \Gamma_{\text{aux}}$ is the duplicate of a vector $\boldsymbol{\alpha}_\ell \in \Gamma^{(k)}$, node ℓ can be removed. Nodes linking to ℓ are redirected to m.
- Similarly, if a vector $\boldsymbol{\alpha}_m \in \Gamma_{\text{aux}}$ is strictly dominated by a vector $\boldsymbol{\alpha}_\ell \in \Gamma^{(k)}$, node ℓ can be removed. Nodes linking to ℓ are redirected to m.
- If a node ℓ has no associated vector in Γ_{aux} and no node that remains in the graph links to it, it can be removed.

Policy iteration for POMDPs is thus summarized in Algorithm 6.3. The policy evaluation step (i) corresponds to line 4; the policy improvement step (ii) corresponds to lines 5 to 15. The node m added in line 10 is constructed according to (6.23), as discussed above. Whenever a node ℓ is removed in line 12, nodes linking to ℓ are redirected to node m. Finally, the function remove_unlinked in line 15 recursively verifies, for each node in $\mathcal{G}^{(k+1)}$ whether it is reachable from a node associated with a vector in Γ_{aux}.

Algorithm 6.3 Policy iteration for POMDPs.

```
Require: POMDP model $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c, \gamma)$; maximum number of
    iterations, $K_{\max}$
 1: Initialize $\pi^{(0)}$ to random policy represented as a finite graph $\mathcal{G}^{(0)}$ with $L$ nodes
 2: $k \leftarrow 0$
 3: repeat
 4:     $\Gamma^{(k)} \leftarrow \{\boldsymbol{\alpha}_\ell \mid \boldsymbol{\alpha}_\ell$ solves (6.22), $\ell = 1, \dots, L\}$
 5:     $\Gamma_{\text{aux}} = \mathsf{incremental\_pruning}(\mathcal{M}, \Gamma^{(k)})$
 6:     $\mathcal{G}^{(k+1)} = \mathcal{G}^{(k)}$
 7:     for $\boldsymbol{\alpha}_m \in \Gamma_{\text{aux}}$ do
 8:         Add node $m$ to $\mathcal{G}^{(k+1)}$
 9:         if $\boldsymbol{\alpha}_m \leq \boldsymbol{\alpha}_\ell$ for $\boldsymbol{\alpha}_\ell \in \Gamma^{(k)}$ then
10:             Remove node $\ell$
11:         end if
12:     end for
13:     $\mathcal{G}^{(k+1)} = \mathsf{remove\_unlinked}(\mathcal{G}^{(k+1)})$
14:     $k \leftarrow k + 1$
15: until $\pi^{(k-1)} = \pi^{(k)}$ or $k > K_{\max}$
16: return $\pi^{(k)}$
```

We can now formulate Proposition 6.6, extending Proposition 5.8 from Chapter 5 to POMDPs. In particular, Proposition 6.6 establishes that the policy improvement step in Algorithm 6.3 yields a policy with an improved cost-to-go function (except if the policy is already optimal).

Proposition 6.6. *Given a POMDP $(\mathcal{X}, \mathcal{A}, Z, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c, \gamma)$, let J^π denote the cost-to-go function associated with some policy, π, represented as a finite policy graph $\mathcal{G}$. If π_g is the policy obtained from π after the policy improvement step of Algorithm 6.3, then*

$$J^{\pi_g}(\boldsymbol{b}) \leq J^\pi(\boldsymbol{b}),$$

for every belief $\boldsymbol{b} \in \mathcal{B}$. Moreover, if $J^{\pi_g}(\boldsymbol{b}) = J^\pi(\boldsymbol{b})$ for every $\boldsymbol{b} \in \mathcal{B}$, then $J^{\pi_g} = J^*$ and $\pi_g = \pi^*$.

Proof. See Exercise 6.2. □

It follows that—much like VI—for any $\varepsilon > 0$, PI converges to a policy π that is ε-optimal, i.e., such that $\|J^* - J^\pi\| \leq \varepsilon$. For the particular case of finitely transient POMDPs, the optimal policy admits an exact representation as a finite state controller, and in such case PI is guaranteed to converge to the optimal policy in a finite number of iterations. We note, however, that finite transience cannot be easily be tested and, as such, is of little practical use.

6.4.3 Examples

We now illustrate the application of VI and PI to the domains from Section 6.2.2.

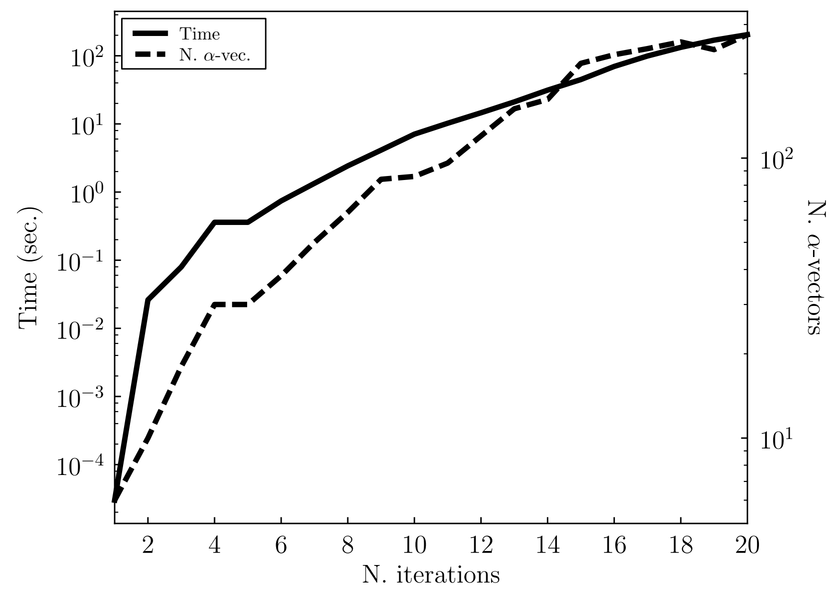


Figure 6.11 Number of computed α-vectors and total computation time for 20 iterations of VI using incremental pruning.

The tiger problem

We saw, in Example 6.3, that the parsimonious representation of the cost-to-go estimate $J^{(1)}$ used 3 vectors, computed directly from the cost function, i.e.,

$$\Gamma^{(1)} = \left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix} \right\}.$$

The parsimonious representation for $J^{(2)}$, on the other hand, already requires 5 α-vectors, as seen in Fig. 6.8(b), where

$$\Gamma^{(2)} = \left\{ \begin{bmatrix} 0.175 \\ 0.175 \end{bmatrix}, \begin{bmatrix} 0.111 \\ 0.276 \end{bmatrix}, \begin{bmatrix} 0.276 \\ 0.111 \end{bmatrix}, \begin{bmatrix} 1.075 \\ 0.075 \end{bmatrix}, \begin{bmatrix} 0.075 \\ 1.075 \end{bmatrix} \right\}.$$

As we proceed with more iterations of VI, the number of α-vectors and the amount of computation time increases as depicted in Fig. 6.11. In particular, after 20 iterations, 244 α-vectors are required to represent $J^{(20)}$, where the last iteration alone takes more than 30 seconds.

As for the resulting function, we depicted $J^{(20)}$ in Fig. 6.12, where we also indicate the partition of the belief induced by $J^{(20)}$ and the corresponding actions. Beliefs $\boldsymbol{b}$ on the leftmost part of the plot (corresponding to high degree of certainty that the tiger is on the right), the prescribed action is $\pi(\boldsymbol{b}) = \text{OL}$. Conversely, beliefs on the rightmost part of the plot (where the tiger is on the right with a high degree of certainty) the prescribed action is $\pi(\boldsymbol{b}) = \text{OR}$. In the remaining belief space, the prescribed action is to listen.

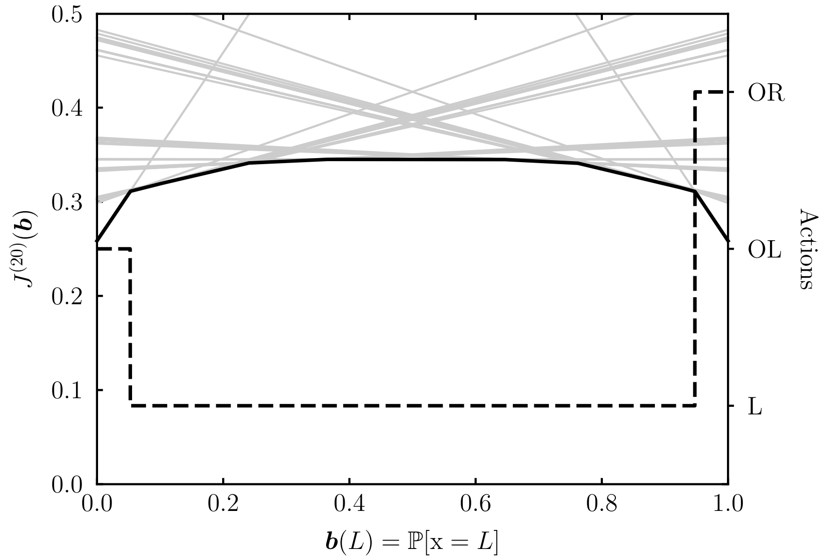


Figure 6.12 Estimate $J^{(20)}$ and corresponding α-vectors. Note that, in spite of the fact that $\Gamma^{(20)} \approx 250$, most such vectors have a very small witness region and contribute very little to the representation of $J^{(20)}$. We also indicate the prescribed action in each area of the belief space.

Finally, it is noteworthy that most α-vectors in $\Gamma^{(20)}$ have a very small witness region and thus add little to the representation of $J^{(20)}$.

Web advertising

We use value and policy iteration on the web advertising problem. Unlike the tiger problem, the policy and the cost-to-go function are difficult to visualize, given that the state space has 4 states and the corresponding belief is, therefore, a 4-dimensional vector. We depict in Fig. 6.13 the policy graph obtained after 3 iterations of PI, where the initial graph has a single node associated with a random action. After 3 iterations, none of the two methods is yet close to converging. Nevertheless, the resulting policy graph suggests several observations.

First of all, we note that the graph contains 2 central nodes labelled with the actions M and W, with a larger number of inbound transitions. This indicates that the policy will spend most of the time in these nodes or, equivalently, that the corresponding α-vectors will cover most of the belief space. There is also a number of peripheral nodes, which cover different initial beliefs and mostly lead to the central nodes. Interestingly, such structure persists as we increase the number of iterations, with a small number of central nodes that concentrate most transitions from a large number of peripheral nodes (see Fig. 6.14).

A second observation is concerned with the actual behavior portrayed in the graph. As seen in page 187, the transition probabilities associated with actions M

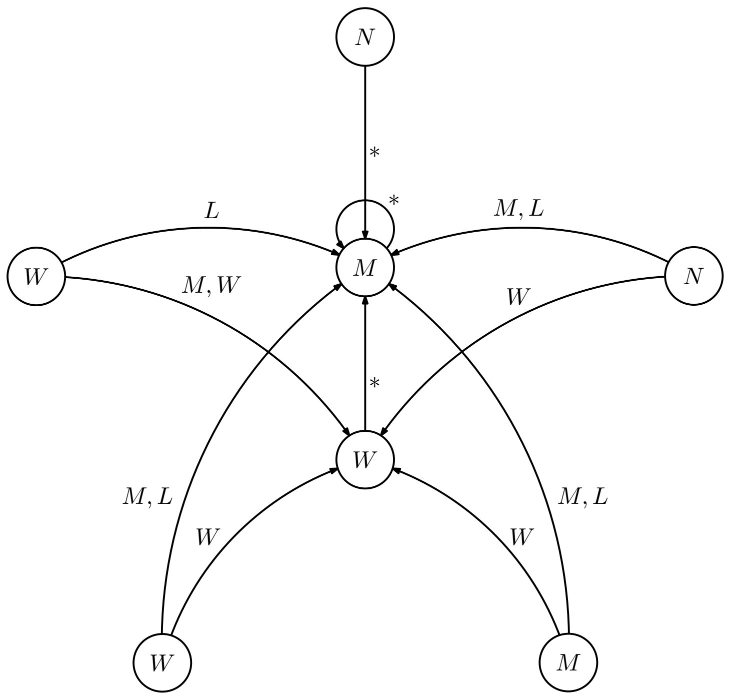


Figure 6.13 Policy graph for the web advertising domain after 3 iterations of PI. The graph includes a total of 7 nodes, and the transitions between the nodes are labeled with the associated observation. A label of $*$ indicates that the transition is associated with all observations.

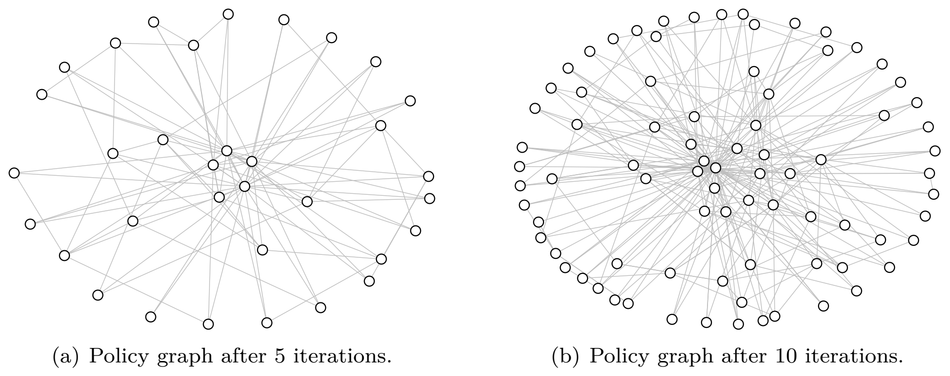

(a) Policy graph after 5 iterations.
(b) Policy graph after 10 iterations.

Figure 6.14 Policy graph representations for the web advertising domain after 5 and 10 iterations of PI. We omitted node and transition labels to avoid cluttering the graphs. However, the structure outlined in the graph of Fig. 6.13 is still perceivable.

and W are given by

$$\boldsymbol{P}_M = \begin{bmatrix} 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.5 & 0.5 \\ 0.05 & 0.15 & 0.8 & 0.0 \\ 0.01 & 0.7 & 0.29 & 0.0 \end{bmatrix}, \qquad \boldsymbol{P}_W = \begin{bmatrix} 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.5 & 0.5 \\ 0.01 & 0.7 & 0.0 & 0.29 \\ 0.10 & 0.15 & 0.0 & 0.75 \end{bmatrix}.$$

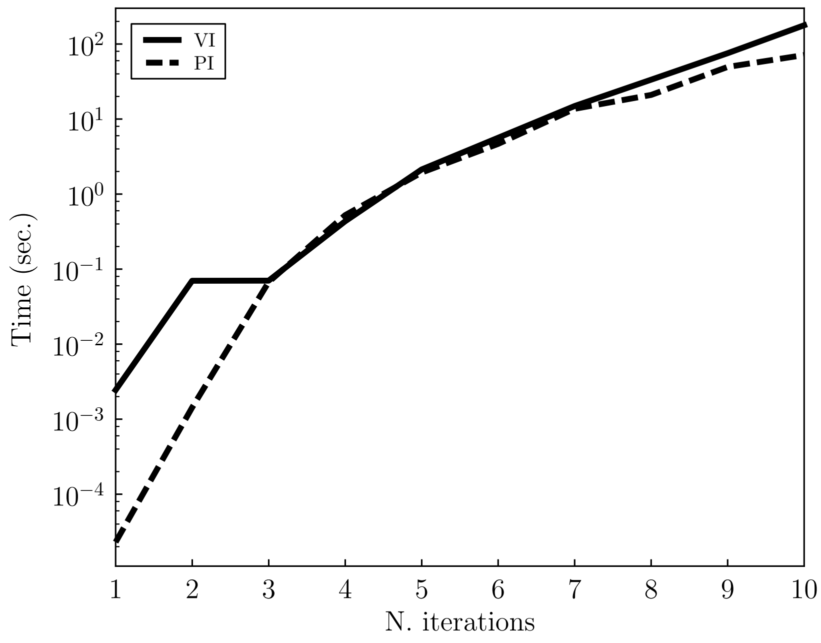
When showed the right ad, none of the two types of customers is likely to leave the site. However, customers interested in mountain sports are less likely to buy (and more likely to remain in the webpage) than customers interested in water sports. Therefore, at any moment (and without any additional information), it is slightly more likely that a visitor is interested in mountain sports than water sports. The central node thus covers those situations in which the system is relatively confident that it is interacting with a customer interested in mountain sports (thus the action M). Conversely, the node immediately below corresponds to the situation where the system is relatively confident that is dealing with a customer interested in water sports. The other nodes correspond to different degrees of certainty regarding the type of customer, and will move to one of these nodes depending on the observations.

As in the tiger example, the number of nodes/α-vectors required to represent the policy/cost-to-go function quickly increases with the number of iterations. After 10 iterations, the number of nodes/α-vectors is above 90, making even the policy graph hard to draw.

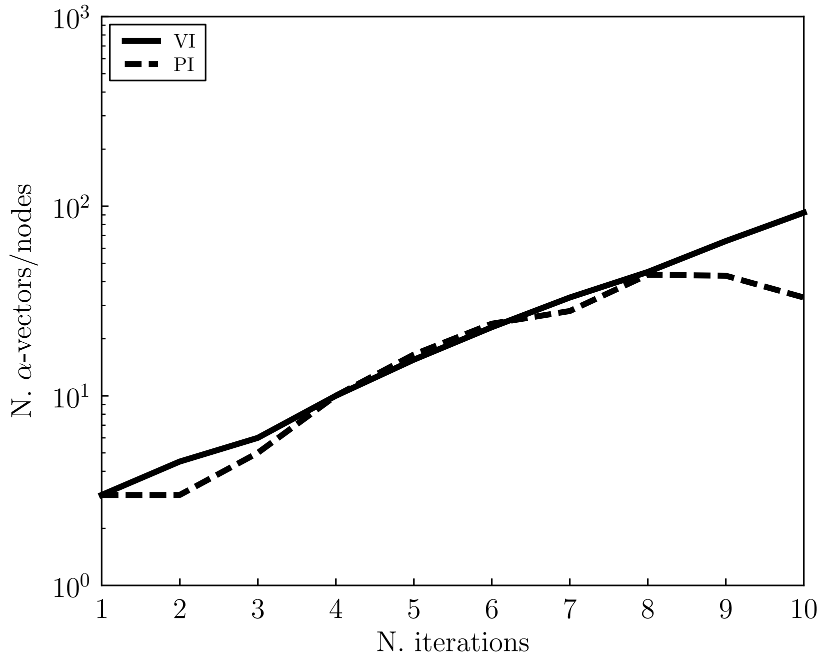
We conclude by comparing, in Fig. 6.15, the computation time taken and number of α-vectors/policy graph nodes generated by both value and policy iteration in the web advertising domain. Both methods perform similarly: the computational time grows exponentially with the number of iterations. The small advantage of VI over PI can easily be explained by noting that the policy evaluation step in PI solves a linear system whose dimension grows with the number of nodes in the policy graph. Therefore, as the number of nodes increases, so does the computational weight of the policy evaluation step. A similar situation was observed in Chapter 5. Likewise, the number of α-vectors used to represent the value function in successive iterations of VI and the number of nodes used to represent the policy in successive iterations of PI also grows exponentially with the number of iterations: after 10 iterations, thus number is already over 80.

Shuttle

As with the previous examples, the computation time and the size of the representation for the shuttle problem quickly grow with the number of iterations. We depict, in Fig. 6.16, the policy graph obtained for the shuttle problem after 3 iterations of PI, where the initial graph has a single node associated with a random action. We observe a similar situation to that observed in the web advertising problem: the graph can be organized around a small number of central nodes to which most other nodes converge. Once again, this suggests that the majority of the decisions of the agent arise from these nodes or, in other words, that the α-vectors corresponding to such central nodes cover a significant part of the belief space.



(a) Total computation time.



(b) N. of α-vectors.

Figure 6.15 Total computation time and number of α-vectors for both VI and PI in the web advertising domain. Both the time and the number of α-vectors grow exponentially in both methods (note the logarithmic scale) with the number of iterations.

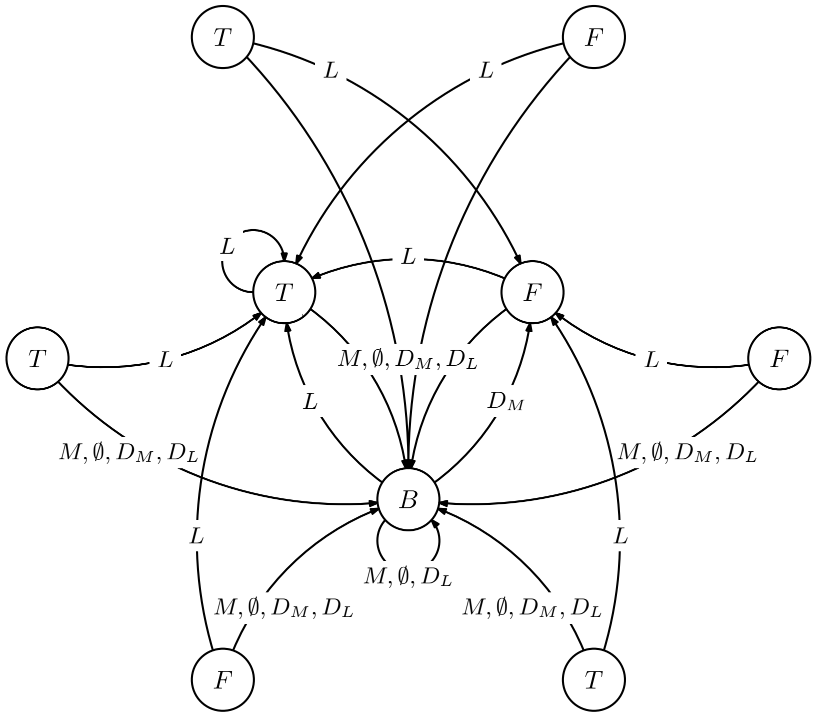


Figure 6.16 Policy graph for the shuttle example. Transitions are labeled with the corresponding observation(s). Missing transitions are self-transitions.

Additionally, while bearing in mind that the graph in Fig. 6.16 is obtained after only 3 iterations of PI and is, therefore, far from the optimal policy, it is still possible to attempt an interpretation of the the corresponding behavior based on a broad inspection of the topology of the graph.

In most nodes, if faced with the least recently visited station (L), will transition to a state where the prescribed action is "turn around" (T). Conversely, when seeing nothing ($\emptyset$)—indicating that the shuttle is in space between the two stations—or the most recently visited station (M) most nodes will transition a node where the action is "backup" (B). Finally, when docked in either station, all nodes transition to the central node, where the shuttle backs up (B).

6.5 Non-exact solution methods

As was apparent from the examples in Section 6.4, both the time required to compute the optimal policy for a POMDP and the memory required to store it quickly become unmanageable even for the smallest problems.[7] Therefore, in order

[7]This observation is made more concrete in Section 6.6, when we discuss the computational complexity associated with solving POMDPs.

to solve moderate to large-sized POMDPs, we need alternative methods that are computationally more attractive than those surveyed so far.

6.5.1 MDP heuristics

One first class of methods takes advantage of the fact that MDPs can be efficiently solvable, using the solution to the underlying MDP to compute a policy for POMDPs. While such approaches offer no guarantees regarding the quality of the resulting policy, they are quite fast to compute and yield, in specific problems, acceptable performance.

Most likely state

Let $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c, \gamma)$ denote an arbitrary POMDP, and let Q_{MDP} and π_{MDP} denote the optimal Q-function and policy for the underlying MDP, respectively. In other words, Q_{MDP} and π_{MDP} verify, for all $x \in \mathcal{X}$ and $a \in \mathcal{A}$,

$$Q_{\text{MDP}}(x, a) = c(x, a) + \gamma \sum_{y \in \mathcal{X}} \boldsymbol{P}(y \mid x, a) \min_{a' \in \mathcal{A}} Q_{\text{MDP}}(y, a'),$$
$$\pi_{\text{MDP}}(x) = \operatorname*{argmin}_{a \in \mathcal{A}} Q_{\text{MDP}}(x, a).$$

The main difficulty posed by partial observability lies on the fact that the state at each time step t, x_t, is unknown and observed only indirectly through z_t. However, the belief $\mathbf{b}_t$ does provide an estimate of the underlying state. Given a belief $\boldsymbol{b} \in \mathcal{B}$, let

$$\hat{x}(\boldsymbol{b}) = \operatorname*{argmax}_{x \in \mathcal{X}} \mathbf{b}(x).$$

The state $\hat{x}(\boldsymbol{b})$ is the most likely state according to $\boldsymbol{b}$. The *most likely state (MLS) heuristic* proposes the policy

$$\pi_{\text{MLS}}(\boldsymbol{b}) \overset{\text{def}}{=} \pi_{\text{MDP}}(\hat{x}(\boldsymbol{b})).$$

The MLS heuristic ignores all state uncertainty and selects, at each time step t, the action prescribed by π_{MDP} in the most likely state at that time step.

Example 6.4 We illustrate the application of the MLS heuristic in the tiger problem from page 186. The optimal policy for the underlying MDP is, trivially,

$$\pi_{\text{MDP}}(L) = \text{OR}, \qquad \text{and} \qquad \pi_{\text{MDP}}(R) = \text{OL}.$$

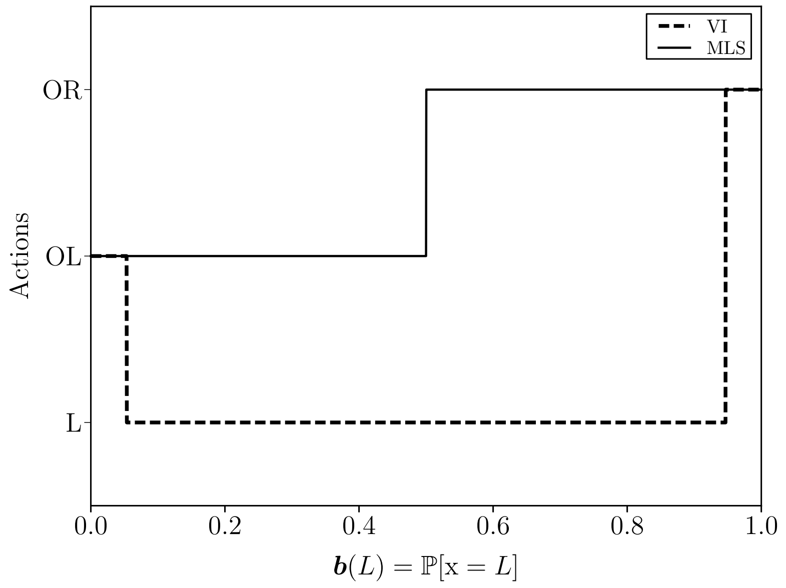


Figure 6.17 Comparison between the policy prescribed by the MLS heuristic and the policy obtained after 20 iterations of VI (as seen in Fig. 6.12).

We get the MLS policy

$$\pi_{\mathrm{MLS}}(\boldsymbol{b}) = \begin{cases} \mathrm{OL} & \text{if } \boldsymbol{b}(L) < \boldsymbol{b}(R) \\ \mathrm{OR} & \text{otherwise.} \end{cases}$$

which is depicted in Fig. 6.17. Since the MDP never uses the "Listen" action (as there is no need to collect information), neither does the MLS heuristic. Therefore, even in situations of great uncertainty—for example, if $\boldsymbol{b}(L) = 0.51$ and $\boldsymbol{b}(R) = 1 - \boldsymbol{b}(L) = 0.49$—the MLS heuristic will still naively select the action associated with the most likely state, disregarding all uncertainty.

Action voting

The MLS heuristic selects its action based solely on the most likely state, ignoring the information that the belief provides regarding all other states. This is particularly inconvenient in beliefs with large entropy, in which the probability mass is distributed evenly by all states.

One possible alternative is to instead consider how the probability mass is distributed over *actions*. Considering that each state $x \in \mathcal{X}$ "votes" on action $\pi_{\mathrm{MDP}}(x)$ with probability $\boldsymbol{b}(x)$, the *action voting heuristic* proposes the policy

$$\pi_{\mathrm{AV}}(\boldsymbol{b}) \overset{\mathrm{def}}{=} \operatorname*{argmax}_{a \in \mathcal{A}} \sum_{x \in \mathcal{X}} \boldsymbol{b}(x) \mathbb{I}\left[\pi_{\mathrm{MDP}}(x) = a\right].$$

Intuitively, the AV heuristic proposes the most consensual action across states, given the belief $\boldsymbol{b}$.

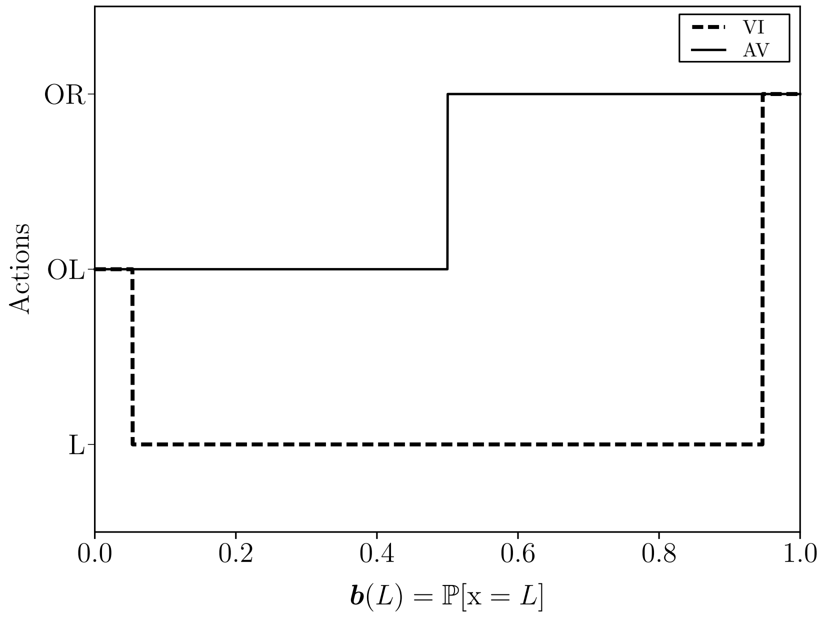


Figure 6.18 Comparison between the policy prescribed by the AV heuristic and the policy obtained after 20 iterations of VI.

Example 6.5 The AV heuristic applied in the tiger problem is, in all aspects, similar to the MLS heuristic, since there are only two states in the underlying MDP. Thus, we get the policy

$$\pi_{\mathrm{AV}}(\boldsymbol{b}) = \begin{cases} \mathrm{OL} & \text{if } \boldsymbol{b}(L) < \boldsymbol{b}(R) \\ \mathrm{OR} & \text{otherwise.} \end{cases}$$

which is depicted in Fig. 6.18. Once again, since the MDP never uses the "Listen" action, neither does the AV heuristic. Therefore, even if the AV heuristic makes better use of the information in $\boldsymbol{b}$, it is still unable to properly handle situations in which the probability mass is evenly distributed across actions.

Q-MDP

Much due to their extreme simplicity, the MLS and AV heuristics are unable to handle belief states in which the probability mass is distributed too evenly either across the states or across the actions. The *Q-MDP heuristic* is, perhaps, the most widely known MDP heuristic, not only because it offers a more robust action selection heuristic than MLS or AV, but also because it is amenable to an intuitive interpretation.

At each time step t, the Q-MDP heuristic considers only the uncertainty in the current state, x_t, and ignores partial observability thereafter. In formal terms, if J_{MDP} is the optimal cost-to-go function for the underlying MDP, the Q-MDP heuristic consists of the approximation

$$\begin{aligned}
J^*(\boldsymbol{b}) &= \min_{a\in\mathcal{A}} \sum_{x\in\mathcal{X}} \boldsymbol{b}(x)\left[c(x,a) + \gamma \sum_{z\in\mathcal{Z}}\sum_{y\in\mathcal{X}} \boldsymbol{P}(y \mid x,a)\boldsymbol{O}(z \mid y,a)J^*(\boldsymbol{B}(\boldsymbol{b},z,a))\right]\\
&\approx \min_{a\in\mathcal{A}} \sum_{x\in\mathcal{X}} \boldsymbol{b}(x)\left[c(x,a) + \gamma \sum_{z\in\mathcal{Z}}\sum_{y\in\mathcal{X}} \boldsymbol{P}(y \mid x,a)\boldsymbol{O}(z \mid y,a)J_{\text{MDP}}(y)\right]\\
&= \min_{a\in\mathcal{A}} \sum_{x\in\mathcal{X}} \boldsymbol{b}(x)\left[c(x,a) + \gamma \sum_{y\in\mathcal{X}} \boldsymbol{P}(y \mid x,a)J_{\text{MDP}}(y)\right]\\
&= \min_{a\in\mathcal{A}} \sum_{x\in\mathcal{X}} \boldsymbol{b}(x)Q_{\text{MDP}}(x,a).
\end{aligned}$$

Therefore, the Q-MDP heuristic proposes the policy

$$\pi_{\text{QMDP}}(\boldsymbol{b}) \overset{\text{def}}{=} \operatorname*{argmin}_{a\in\mathcal{A}} \sum_{x\in\mathcal{X}} \boldsymbol{b}(x)Q_{\text{MDP}}(x,a). \tag{6.24}$$

Example 6.6 We now illustrate the application of the Q-MDP heuristic in the tiger problem. In the underlying MDP, the optimal policy is always able to select the action with cost 0, for which reason the optimal Q-function comes

$$\boldsymbol{Q}_{\text{MDP}} = \begin{bmatrix} 1 & 0 & 0.1 \\ 0 & 1 & 0.1 \end{bmatrix}.$$

The Q-MDP heuristic thus prescribes the action OL whenever $\boldsymbol{b}(L) < 0.1$. Similarly, it prescribes action OR whenever $\boldsymbol{b}(R) < 0.1$ or, equivalently, $\boldsymbol{b}(L) > 0.9$. For $0.1 \leq \boldsymbol{b}(L) \leq 0.9$ it prescribes action L. The resulting policy is depicted in Fig. 6.19 and can be summarized as

$$\pi_{\text{MDP}}(\boldsymbol{b}) = \begin{cases} \text{OL} & \text{if } \boldsymbol{b}(L) < 0.1\\ \text{OR} & \text{if } \boldsymbol{b}(L) > 0.9\\ \text{L} & \text{otherwise.} \end{cases}$$

Note that, unlike the MLS and AV heuristics, Q-MDP does select the "Listen" action, even if it is never used in the underlying MDP. The Q-MDP policy is, however, more confident than the VI policy. In other words, the Q-MDP policy will open a door sooner than the VI policy, which is in accordance with the fact that Q-MDP ignores partial observability in the future.

The next example illustrates how the assumptions behind Q-MDP can, in certain situations, lead to poor action selection.

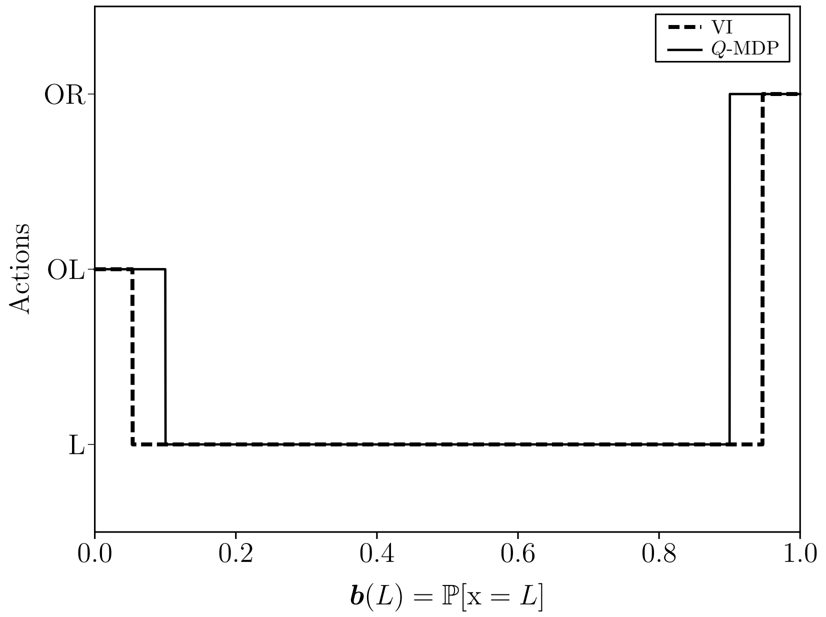


Figure 6.19 Comparison between the policy prescribed by the Q-MDP heuristic and the policy obtained after 20 iterations of VI. Note that the two policies are qualitatively similar, disagreeing only on where the agent should switch from listening to one of the door opening actions.

Example 6.7 Consider the decision problem described by the transition diagram in Fig. 6.20. In the initial state, I, independently of the action of the agent, the system moves to one of the two states A_1 or A_2 with equal probability. The agent makes the same observation (A) in both states and, therefore, knows not in which of the two states the system is. All other transitions are deterministic.

At each time step, the agent can select one of three possible actions, a, b, c. All actions are equivalent in all states except A_1 and A_2, where each action leads to a different state. Finally, all actions have a cost of 1 in all states, except in state D, where all actions have a cost of 0.

Let $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c, \gamma)$ denote the POMDP modeling the decision problem just described, where:

- $\mathcal{X} = \{I, A_1, A_2, B, C, D, E\}$;
- $\mathcal{A} = \{a, b, c\}$;
- $\mathcal{Z} = \{I, A, B, C, D, E\}$;

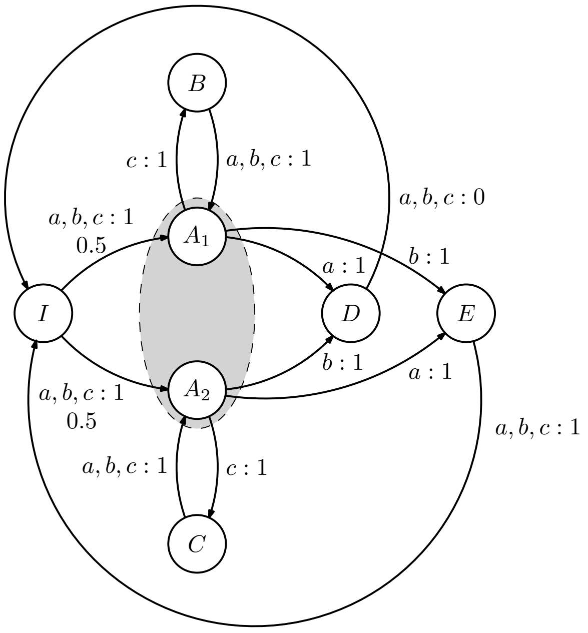


Figure 6.20 7-state POMDP in which partial observability occurs only in states A_1 and A_2. All transitions have a cost of 1, except the transition from state D to state I.

- The transition probabilities are given by

$$\boldsymbol{P}_a = \begin{bmatrix} 0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad \boldsymbol{P}_b = \begin{bmatrix} 0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

and

$$\boldsymbol{P}_c = \begin{bmatrix} 0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

- The observation probabilities are given by

$$\boldsymbol{O}_a = \boldsymbol{O}_b = \boldsymbol{O}_c = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

- The cost function can be summarized as the matrix

$$\boldsymbol{C} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}.$$

- Finally, we use $\gamma = 0.95$.

Computing the optimal Q-function for the underlying MDP yields

$$\boldsymbol{Q}_{\text{MDP}} = \begin{bmatrix} 13.67 & 13.67 & 13.67 \\ 13.34 & 14.29 & 13.99 \\ 14.29 & 13.34 & 13.99 \\ 13.67 & 13.67 & 13.67 \\ 13.67 & 13.67 & 13.67 \\ 12.99 & 12.99 & 12.99 \\ 13.99 & 13.99 & 13.99 \end{bmatrix}.$$

Suppose that, at some time step t, $\mathbf{b}_t = \begin{bmatrix} 0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 \end{bmatrix}$. Then,

$$\mathbf{b}_t \cdot \boldsymbol{Q}_{\text{MDP}} = \begin{bmatrix} 13.81 & 13.81 & 13.99 \end{bmatrix},$$

and Q-MDP prescribes any of the actions a or b. In contrast, running 20 iterations of VI for the POMDP $\mathcal{M}$ yields the Q-values

$$\boldsymbol{Q}^*(\mathbf{b}_t) = \begin{bmatrix} 16.28 & 16.28 & 16.21 \end{bmatrix},$$

showing that—as we would expect—the POMDP policy actually selects the action c in the same situation.

The difference between the two policies can easily be interpreted in light of the fact that Q-MDP disregards partial observability in future time steps. If the state becomes fully observable from the next time step onward, the number of steps necessary to disambiguate between A_1 and A_2 (by selecting action c) are more costly than simply choosing one of the two actions a or b and eventually paying the cost in E. Conversely, the VI policy realizes that, by taking action c, it is possible to avoid with full certainty the cost in state E, which actually leads to superior performance.

Fast informed bound

As illustrated in Examples 6.4 through 6.7, all MDP heuristics so far rely too heavily on the underlying MDP solutions, which makes them perform poorly, especially in situations that require the agent to explicitly collect information regarding the state (as in Example 6.7). The *fast informed bound* (FIB) heuristic does consider the impact of partial observability on the decision process of the agent even if, strictly speaking, it does not build directly on the MDP solution. Nevertheless, it retains the simplicity of the MDP heuristics.

To introduce the *fast informed bound* (FIB) heuristic, we start by comparing the optimal cost-to-go J^* and its estimate according to the Q-MDP heuristic. As seen in Section 6.4, we can write

$$J^*(\boldsymbol{b}) = \min_{a\in\mathcal{A}} \left[\sum_{x\in\mathcal{X}} \boldsymbol{b}(x)c(x,a) + \gamma \sum_{z\in\mathcal{Z}} \min_{\boldsymbol{\alpha}\in\Gamma^*} \sum_{x,y\in\mathcal{X}} \boldsymbol{b}(x)\boldsymbol{P}(y \mid x,a)\boldsymbol{O}(z \mid y,a)\boldsymbol{\alpha}_y\right]. \tag{6.25}$$

where Γ^* is the (possibly infinite) optimal set of α-vectors that represent J^*. The Q-MDP heuristic, on the other hand, relies on the estimate

$$J_{\text{QMDP}}(\boldsymbol{b}) = \min_{a\in\mathcal{A}} \sum_{x\in\mathcal{X}} \boldsymbol{b}(x) \left[c(x,a) + \gamma \sum_{y\in\mathcal{X}} \boldsymbol{P}(y \mid x,a) \min_{\boldsymbol{\alpha}\in\Gamma_{\text{QMDP}}} \boldsymbol{\alpha}_y\right] \tag{6.26}$$

where there is precisely one α-vector $\boldsymbol{\alpha}_a \in \Gamma_{\text{QMDP}}$ for each action $a \in \mathcal{A}$ with xth component given by

$$[\boldsymbol{\alpha}_a]_x = Q_{\text{MDP}}(x,a).$$

We can equivalently rewrite (6.26) as

$$\begin{aligned} &J_{\text{QMDP}}(\boldsymbol{b}) \\ &= \min_{a\in\mathcal{A}} \left[\sum_{x\in\mathcal{X}} \boldsymbol{b}(x)c(x,a) + \gamma \sum_{z\in\mathcal{Z}} \sum_{x,y\in\mathcal{X}} \boldsymbol{b}(x)\boldsymbol{P}(y \mid x,a)\boldsymbol{O}(z \mid y,a) \min_{\boldsymbol{\alpha}\in\Gamma_{\text{QMDP}}} \boldsymbol{\alpha}_y\right]. \end{aligned} \tag{6.27}$$

The difference between (6.25) and (6.27) lies on where the minimization w.r.t. the α-vectors is performed. In (6.25), and depending on current belief $\boldsymbol{b}$, a single α-vector is selected for each observation $z \in \mathcal{Z}$. In (6.27), in contrast, an α-vector is selected for each next state y, but which is independent both of the current belief, $\boldsymbol{b}$, and the observation.

The FIB heuristic can be seen as a middle ground between (6.25) and (6.27). Like Q-MDP, it does not depend on the current belief $\boldsymbol{b}$. However, like (6.25), it selects a different α-vector for each observation $z \in \mathcal{Z}$. The resulting cost-to-go function can be written as

$$J_{\text{FIB}}(\boldsymbol{b}) \\ = \min_{a \in \mathcal{A}} \left[\sum_{x \in \mathcal{X}} \boldsymbol{b}(x) c(x,a) + \gamma \sum_{z \in \mathcal{Z}} \sum_{x \in \mathcal{X}} \boldsymbol{b}(x) \min_{\boldsymbol{\alpha} \in \Gamma_{\text{FIB}}} \sum_{y \in \mathcal{X}} \boldsymbol{P}(y \mid x,a) \boldsymbol{O}(z \mid y,a) \boldsymbol{\alpha}_y \right]. \tag{6.28}$$

In the set Γ_{FIB} there is exactly one vector $\boldsymbol{\alpha}_a$ for each action $a \in \mathcal{A}$ with xth component given by

$$[\boldsymbol{\alpha}_a]_x = c(x,a) + \gamma \sum_{z \in \mathcal{Z}} \min_{\boldsymbol{\alpha} \in \Gamma_{\text{FIB}}} \sum_{y \in \mathcal{X}} \boldsymbol{P}(y \mid x,a) \boldsymbol{O}(z \mid y,a) \boldsymbol{\alpha}_y.$$

Alternatively, we can interpret FIB as computing a modified MDP Q-function that verifies the recursive relation

$$Q_{\text{FIB}}(x,a) = c(x,a) + \gamma \sum_{z \in \mathcal{Z}} \min_{a' \in \mathcal{A}} \sum_{y \in \mathcal{X}} \boldsymbol{P}(y \mid x,a) \boldsymbol{O}(z \mid y,a) Q_{\text{FIB}}(y,a'). \tag{6.29}$$

The Q-function Q_{FIB} can be computed by adapting VI (Algorithm 5.3 in Chapter 5) in a straightforward manner—simply replacing the operator H by the recursion in (6.29). Moreover, unlike Q-MDP, FIB is able to more successfully accommodate the impact of partial observability in the decision process of the agent. In fact, inspecting (6.25), (6.27) and (6.28), we can immediately conclude that

$$J_{\text{QMDP}} \leq J_{\text{FIB}} \leq J^*,$$

meaning that FIB is provides, in general, a better approximation to J^* than Q-MDP.

Example 6.8 We illustrate the application of the FIB heuristic in the tiger problem. Using the modified version of VI to compute the Q-values in (6.29), we get

$$\boldsymbol{Q}_{\text{FIB}} = \begin{bmatrix} 1.171 & 0.171 & 0.229 \\ 0.171 & 1.171 & 0.229 \end{bmatrix}.$$

FIB thus prescribes the action OL whenever $\boldsymbol{b}(L) < 0.058$. Similarly, it prescribes action OR whenever $\boldsymbol{b}(R) < 0.058$ or, equivalently, $\boldsymbol{b}(L) > 0.942$. For $0.058 \leq \boldsymbol{b}(L) \leq 0.942$ it prescribes action L. The resulting policy is depicted in Fig. 6.21 and can be summarized as

$$\pi_{\text{FIB}}(\boldsymbol{b}) = \begin{cases} \text{OL} & \text{if } \boldsymbol{b}(L) < 0.058 \\ \text{OR} & \text{if } \boldsymbol{b}(L) > 0.942 \\ \text{L} & \text{otherwise.} \end{cases}$$

Much like Q-MDP, FIB also selects the "Listen" action in a large part of the belief space. However, unlike the former, FIB does not assume that partial

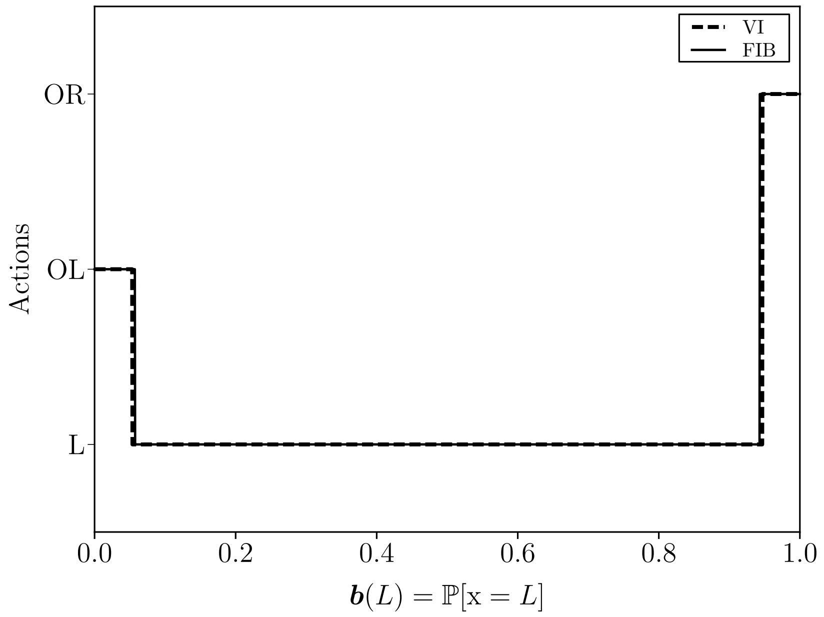


Figure 6.21 Comparison between the policy prescribed by the FIB heuristic and the policy obtained after 20 iterations of VI.

observability will disappear in the future and is, therefore, more conservative than Q-MDP, yielding a near-optimal policy in this example.

Example 6.9 Let us revisit the decision problem described by the transition diagram in Fig. 6.20 and discussed in Example 6.7. Computing the Q-values in (6.29), we now get

$$\boldsymbol{Q}_{\text{FIB}} = \begin{bmatrix} 16.40 & 16.40 & 16.40 \\ 15.80 & 16.75 & 16.21 \\ 16.75 & 15.80 & 16.21 \\ 16.01 & 16.01 & 16.01 \\ 16.01 & 16.01 & 16.01 \\ 15.58 & 15.58 & 15.58 \\ 16.58 & 16.58 & 16.58 \end{bmatrix}.$$

For $\mathbf{b} = \begin{bmatrix} 0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 \end{bmatrix}$, we now get

$$\mathbf{b}_t \cdot \boldsymbol{Q}_{\text{FIB}} = \begin{bmatrix} 16.28 & 16.28 & 16.21 \end{bmatrix},$$

which, as seen in Example 6.7, match the values computed using VI. The FIB heuristic thus selects action c, successfully avoiding the cost in state E and outperforming Q-MDP.

$\diamond$

We conclude by pointing out that more sophisticated heuristics have been explored in the literature (see the discussion in Section 6.7). However, none have any performance guarantees, which makes their use in many practical applications somewhat of a gamble. In the continuation, we discuss a class of methods derived from those in Section 6.4. These approximate methods successfully overcome the computational limitations of exact solution methods, while building on their underlying principles and retaining, therefore, some of their properties.

6.5.2 Approximate dynamic programming

One aspect of exact solution methods that becomes apparent from the examples in Section 6.4 is that they are computationally expensive both in terms of time — iterations become more time consuming as the algorithm progresses—and space— the representations of the cost-to-go functions and policies increase in size as the algorithm progresses. The two aspects are closely interrelated: as the size of the representation grows, so does the time needed to process it in each iteration of the methods.

However, we observed in the tiger example (page 186) that many of the α-vectors used to represent $J^{(k)}$ actually had a small witness region. In other words, if one such vector is ignored, the error incurred in terms of representation of $J^{(k)}$ is negligible. Similarly, we noted in the shuttle example (page 215) that several nodes in the policy graph actually cover "impossible" situations that the agent will never meet in practice.

Since both VI and PI are designed to provide a policy for all possible situations that the agent may encounter, such α-vectors/nodes are necessary. However, in practice, such representations can actually be simplified with minimum loss in performance.

Bearing such ideas in mind, we now go over two classes of methods that enforce a limit in the size of the representation for POMDP solution methods. We first survey *point-based methods*, which apply this fundamental idea to VI, limiting the number of α-vectors used to represent the cost-to-go function. We then discuss *bounded finite-state controllers*, which apply the same fundamental idea to PI, limiting the number of nodes used to represent a POMDP policy.

Point-based value iteration

Point-based methods approximate value iteration by keeping the number of α-vectors used to represent the cost-to-go function at each iteration k, $J^{(k)}$, forcefully limited. Only those α-vectors which are most relevant to represent $J^{(k)}$ are retained. The rationale behind such methods is that the error incurred by discarding those vectors is negligible, if the set of α-vectors retained is sufficiently rich.

In order to determine which α-vectors are relevant, point-based value iteration selects, at each iteration k, a discrete set of belief points, $\mathcal{B}^{(k)} \subset \mathcal{B}$. The cost-to-go function, $J^{(k)}$, is represented exactly at the points in $\mathcal{B}^{(k)}$; for belief points $\boldsymbol{b} \notin \mathcal{B}^{(k)}$, the value of $J^{(k)}(\boldsymbol{b})$ is computed as a linear interpolation of the values of $J^{(k)}$ at the points in $\mathcal{B}^{(k)}$.

The general outline of point-based value iteration (PBVI) is summarized in Algorithm 6.4. At each iteration k, PBVI determines the set of belief points $\mathcal{B}^{(k)}$ used to select the relevant α-vectors. Then, for each belief $\boldsymbol{b} \in \mathcal{B}^{(k)}$, the algorithm computes the corresponding optimal α-vector, i.e., the vector $\boldsymbol{\alpha}^*(\boldsymbol{b})$ that minimizes (6.20). The iteration terminates once superfluous vectors are pruned.

Algorithm 6.4 Point-based value iteration.

```
Require: POMDP model $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c, \gamma)$; tolerance $\varepsilon > 0$;
 1: $k \leftarrow 0$
 2: $\Gamma^{(0)}, \mathcal{B}^{(0)} \leftarrow \mathsf{init}(\mathcal{M})$
 3: repeat
 4:     $\Gamma^{(k+1)} \leftarrow \emptyset$
 5:     $\mathcal{B}^{(k+1)} \leftarrow \mathsf{sample}(\mathcal{M}, \mathcal{B}^{(k)})$
 6:     for all $\boldsymbol{b} \in \mathcal{B}^{(k+1)}$ do
 7:         $\Gamma^{(k+1)} \leftarrow \mathsf{update}(\Gamma^{(k+1)}, \boldsymbol{\alpha}^*(\boldsymbol{b}))$
 8:     end for
 9:     $k \leftarrow k + 1$
10: until $\left\| J^{(k-1)} - J^{(k)} \right\| < \varepsilon$
11: return $\Gamma^{(k)}$
```

Particular instantiations of PBVI differ in:

- The initialization of $\Gamma^{(0)}$ and $\mathcal{B}^{(0)}$ (line 2);
- The selection of the subset $\mathcal{B}^{(k)} \subset \mathcal{B}$ (line 5);
- The update of $\Gamma^{(k)}$ (line 7).

We describe one particular variation of PBVI called PERSEUS and refer to Section 6.7 for a brief overview of some other variations. In PERSEUS, $\mathcal{B}^{(0)}$ is constructed from trajectories of the POMDP generated by following a random exploration policy. The resulting set is held fixed across iterations, i.e., $\mathcal{B}^{(k+1)} = \mathcal{B}^{(k)}, k > 0$. The update step adds new α-vectors to $\Gamma^{(k)}$ only if they lead to a strictly better cost-to-go function. In other words, for each visited belief point $\boldsymbol{b} \in \mathcal{B}^{(k)}$, $\boldsymbol{\alpha}^*(\boldsymbol{b})$ is added to $\Gamma^{(k)}$ only if

$$J^{(k-1)}(\boldsymbol{b}) > \boldsymbol{b} \cdot \boldsymbol{\alpha}^*(\boldsymbol{b}).$$

Moreover, when one such vector is added, any unvisited belief points $\boldsymbol{b}' \in \mathcal{B}^{(k)}$ for which

$$J^{(k-1)}(\boldsymbol{b}') > \boldsymbol{b}' \cdot \boldsymbol{\alpha}^*(\boldsymbol{b})$$

are ignored in the present iteration. In spite of its simplicity, PERSEUS is a powerful method, providing a good tradeoff between computational effort and performance.

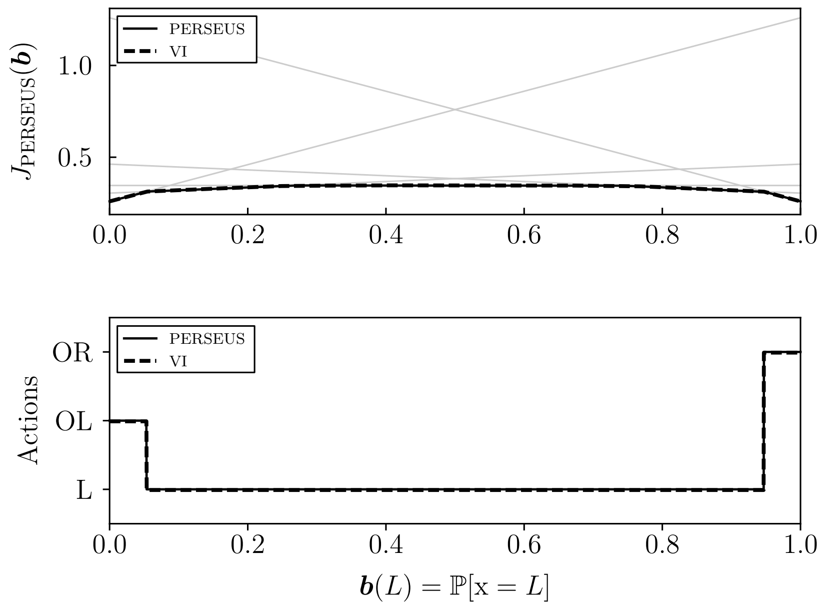


Figure 6.22 Comparison between the cost-to-go function and policy computed using PERSEUS and the policy obtained after 20 iterations of VI.

Example 6.10 Running PERSEUS on the tiger problem yields the cost-to-go function and policy depicted in Fig. 6.22. We note immediately that the cost-to-go function computed by PERSEUS is, for all purposes, indistinguishable from the one obtained after 20 iterations of exact value iteration. However, the latter took around 4 minutes to compute and resulted in a total of 318 α-vectors. PERSEUS, on the other hand, completed in 78 iterations that took a total of 193 ms to complete. The resulting cost-to-go function requires only 5 α-vectors to represent.

Additionally, the policy computed by PERSEUS is also indistinguishable from that computed with VI (see Fig. 6.22). In this particular example, point based methods offer a very appealing tradeoff between computational effort and performance.

Example 6.11 We applied PERSEUS to the slotted ALOHA example from Section 6.4.3, where we set the packet arrival rate to $\lambda = 0.9$ and the maximum number of backlogged packets to $N_{\text{max}} = 9$. In other words, we have a total of 30 states and 9 actions, corresponding to setting the transmission probabilities to $p = 1$, $p = \frac{1}{2}$, up to $p = \frac{1}{9}$.

To observe the impact of the number of sampled beliefs in the performance of PERSEUS, Fig. 6.23 compares the time and number of α-vectors used in the representation of the cost-to-go function as a function of the number of

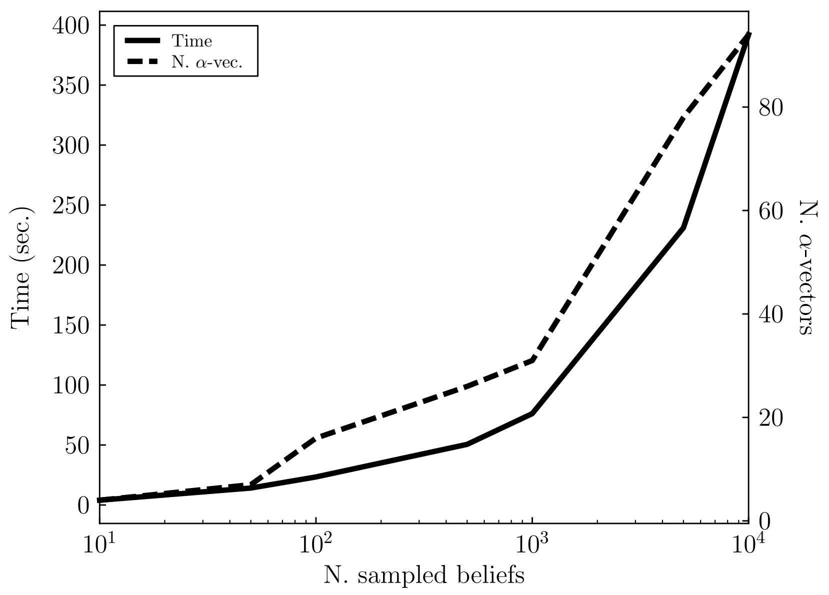
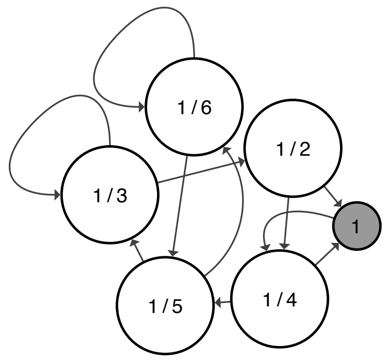


Figure 6.23 Computation time and number of α-vectors used to represent J as a function of the number of sampled beliefs in PERSEUS, for the slotted ALOHA example. The dependence is approximately linear (note the logarithmic scale in the horizontal axis).

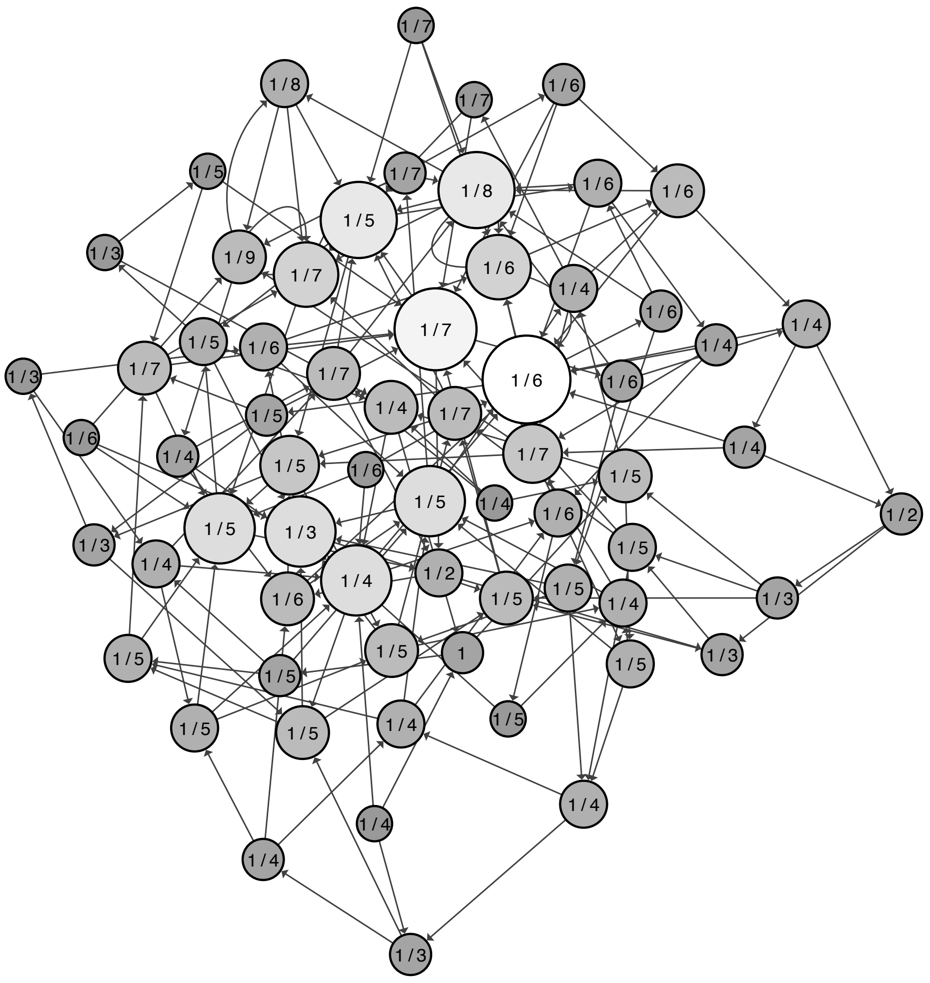
sampled beliefs. As expected, both the time and the size of the representation grow graciously as the number of sampled beliefs increases.

Finally, we depict in Fig. 6.24 the policy graphs obtained by sampling 10 and $5,000$ beliefs. In both policy graphs, the size of the nodes was depicted proportionally to the node's in-degree—i.e., how incoming transition it has. Intuitively, nodes with large in-degrees correspond to more likely actions. As expected, the complexity of the policy increases as the number of sampled beliefs increases. With 10 sampled beliefs only, PERSEUS computes a very simple policy represented with only 5 nodes. Interestingly, the transmission probability is set to 1 only infrequently. When sampling a total of $5,000$ belief points, PERSEUS finished after $6,414$ iterations, taking a total of 2 minutes and 58 seconds and producing a representation for J^* with 65 α-vectors. Although the complexity of the policy graph makes its interpretation difficult, it is nevertheless possible to perceive how the different probabilities are selected. For example, there is a single node associated with the action $a = 1$, with small in-degree. On the other hand, there is a number of nodes associated with the actions $a = \frac{1}{6}$ and $a = \frac{1}{7}$, which also correspond to the two actions in the larger nodes.

An appealing aspect of point-based value iteration—besides its simplicity and computational efficiency—is that we can provide theoretical bounds on its performance. The bounds rely on the intuition that if the set of beliefs used by PVBI



(a) 10 sampled beliefs.



(b) $1,000$ sampled beliefs.

Figure 6.24 Policy graph for the slotted ALOHA problem for different number of sampled beliefs. Node size and shading are proportional to the in-degree (lighter and larger nodes have larger in-degree).

covers extensively the "interesting parts" of $\mathcal{B}$, then hopefully the cost-to-go function computed by PBVI provides a good approximation to the true cost-to-go function.

Let $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c, \gamma)$ denote a POMDP with initial distribution/belief $\boldsymbol{b}_0$. A belief $\boldsymbol{b} \in \mathcal{B}$ is *reachable* from $\boldsymbol{b}_0$ if there is a finite history $h \in \mathcal{H}$ such that

$$\boldsymbol{b}(x) = \mathbb{P}\left[\mathrm{x}_t = x \mid \mathrm{h}_t = h, \mathrm{x}_0 \sim \boldsymbol{b}_0\right].$$

In other words, a belief $\boldsymbol{b}$ is reachable from $\boldsymbol{b}_0$ if there is a history h that leads from $\boldsymbol{b}_0$ to $\boldsymbol{b}$ using the standard belief update in (6.6). The set(s) $\mathcal{B}_{\text{sample}} \subset \mathcal{B}$ used in the different variations of PBVI are constructed by simulating POMDP trajectories and collecting the resulting beliefs. Therefore, all points $\boldsymbol{b} \in \mathcal{B}_{\text{sample}}$ are reachable from $\boldsymbol{b}_0$.

Now given a belief $\boldsymbol{b} \notin \mathcal{B}_{\text{sample}}$, we consider the distance between $\boldsymbol{b}$ and $\mathcal{B}_{\text{sample}}$ as

$$d(\boldsymbol{b}, \mathcal{B}) = \min_{\boldsymbol{b}' \in \mathcal{B}_{\text{sample}}} \|\boldsymbol{b} - \boldsymbol{b}'\|_1 \overset{\text{def}}{=} \min_{\boldsymbol{b}' \in \mathcal{B}_{\text{sample}}} \sum_{x \in \mathcal{X}} |\boldsymbol{b}(x) - \boldsymbol{b}'(x)|.$$

The distance $d(\boldsymbol{b}, \mathcal{B}_{\text{sample}})$ measures how close $\boldsymbol{b}$ is to $\mathcal{B}_{\text{sample}}$. Let $\varepsilon(\mathcal{B}_{\text{sample}})$ denote the maximum distance between any belief and $\mathcal{B}_{\text{sample}}$, i.e.,

$$\varepsilon(\mathcal{B}_{\text{sample}}) = \max_{\boldsymbol{b} \in \mathcal{B}} d(\boldsymbol{b}, \mathcal{B}_{\text{sample}}),$$

we get the following result.

Proposition 6.7. *Given a POMDP $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c, \gamma)$ and a finite set of beliefs $\mathcal{B}_{\text{sample}} \subset \mathcal{B}$ then, for any $k > 0$,*

$$\left\|J_{\text{VI}}^{(k+1)} - J_{\text{PBVI}}^{(k+1)}\right\|_\infty \leq \frac{1}{1-\gamma} \sum_{\kappa=0}^{k} \varepsilon(\mathcal{B}_{\text{sample}}),$$

where $J^(k)_{\text{VI}}$ is the cost-to-go estimate after k iterations of exact VI, and $J_{\text{PBVI}}^{(k)}$ is the cost-to-go estimate after k iterations of PBVI, where each iteration updates the α-vectors for all belief points in $\mathcal{B}_{\text{sample}}$.

Proof. See Section 6.8. □

Proposition 6.7 shows that the error incurred by using PBVI instead of an exact VI update degrades gracefully with $\varepsilon(\mathcal{B}_{\text{sample}})$, which quantifies precisely how well $\mathcal{B}_{\text{sample}}$ covers the relevant parts of $\mathcal{B}$ (those that can be reached from $\boldsymbol{b}_0$). Then, combining Proposition 6.7 with the bound in (5.22) (page 134), we get the following corollary, whose proof is left as an exercise.

Corollary 6.8. *Given a POMDP $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c, \gamma)$, let $\pi_{\text{PBVI}}^{(k)}$ denote the greedy policy w.r.t. $J_{\text{PBVI}}^{(k)}$, where $J_{\text{PBVI}}^{(k)}$ is the cost-to-go function obtained* *after k iterations of PBVI using some finite set of beliefs $\mathcal{B}_{\text{sample}} \subset \mathcal{B}$. Then,*

$$\lim_{k \to \inf} \left\| J^{\pi^{(k)}_{\text{PBVI}}} - J^* \right\|_\infty \leq \frac{2\gamma}{(1-\gamma)^2} \varepsilon(\mathcal{B}_{\text{sample}}).$$

Proof. See Exercise 6.4. □

We conclude by emphasizing that the above results consider that PBVI computes the α-vectors for every belief point $\boldsymbol{b} \in \mathcal{B}_{\text{sample}}$, which is not guaranteed in all PBVI algorithms (for example *perseus*). Additionally, stronger results can be established by further restricting the process by which the beliefs in $\mathcal{B}_{\text{sample}}$, so as to ensure that $\varepsilon(\mathcal{B}_{\text{sample}})$ is small (see discussion in Section 6.7).

Bounded policy iteration

Another class of approximate methods modify the exact policy iteration algorithm described in Section 6.4.2 by forcefully bounding the size of the policy representation at each iteration of PI. As with approximate value iteration, the intuition is that most nodes in the policy graph computed by PI at each iteration can safely be discarded without incurring a significant loss in performance.

Recall that PI maintains, at each iteration k, a representation $\mathcal{G}^{(k)}$ of a policy as a policy graph/finite state controller (FSC). Policy evaluation is performed by computing, for each node in $\mathcal{G}^{(k)}$, a corresponding α-vector. The resulting set of α-vectors is then improved using a standard VI update, and a new FSC $\mathcal{G}^{(k+1)}$ is then derived from the new set of α-vectors. The growth in the size of $\mathcal{G}^{(k)}$ is due to the increase in the number of α-vectors that takes place in the VI step.

We can easily circumvent this growth in the number of α-vectors by reusing the ideas from PBVI in the context of PI. In particular, if we replace the VI step in PI by a PBVI step, we can readily ensure that the number of new α-vectors (and hence nodes in the FSC representation) remains bounded. Unsurprisingly, such approach is known as *point-based policy iteration*, and is summarized in Algorithm 6.5.

PBPI is essentially identical to standard PI (Algorithm 6.3), with single difference that the exact VI step in Algorithm 6.3 is replaced by a PBVI step. The following result is an immediate consequence of Propositions 6.6 and 6.7.

Proposition 6.9. *Given a POMDP $(\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c, \gamma)$, let $\pi^{(k)}$ denote the policy computed after the kth iteration of Algorithm 6.5, and $\mathcal{B}_{\text{sample}}$ the set of belief points used for the PBVI step. Then,*

$$J^{\pi^{(k+1)}}(\boldsymbol{b}) \leq J^{\pi^{(k)}}(\boldsymbol{b}),$$

for all $\boldsymbol{b} \in \mathcal{B}_{\text{sample}}$, where the inequality is strict for at least one point.

Proof. See Exercise 6.5. □

Algorithm 6.5 Point-based policy iteration.

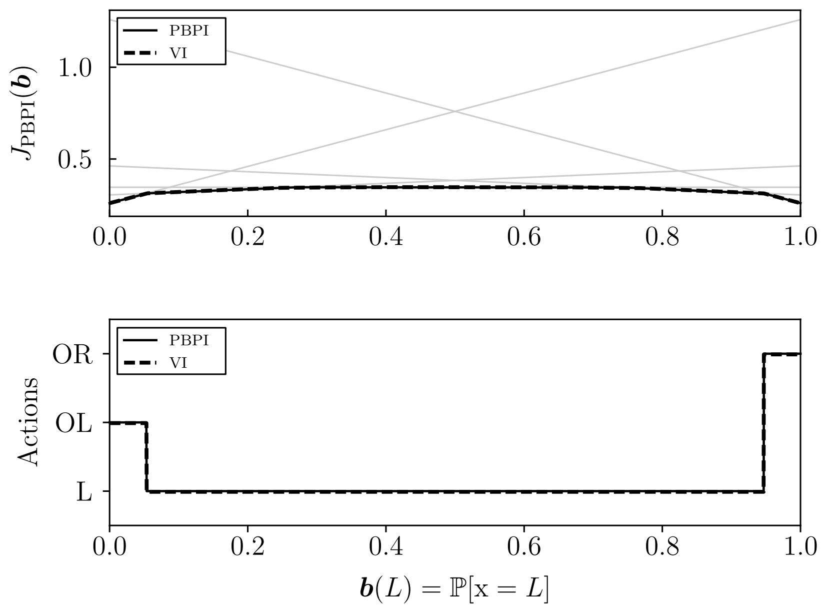
```
Require: POMDP model $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c, \gamma)$; maximum number of
    iterations, $K_{\max}$
 1: Initialize $\pi^{(0)}$ to random policy represented as a finite graph $\mathcal{G}^{(0)}$ with $L$ nodes
 2: $\mathcal{B}^{(0)} \leftarrow \mathsf{init}(\mathcal{M})$
 3: $k \leftarrow 0$
 4: repeat
 5:     $\Gamma^{(k)} \leftarrow \{\boldsymbol{\alpha}_\ell \mid \boldsymbol{\alpha}_\ell \text{ solves } (6.22), \ell = 1, \ldots, L\}$
 6:     $\mathcal{B}^{(k+1)} \leftarrow \mathsf{sample}(\mathcal{M}, \mathcal{B}^{(k)}$
 7:     $\Gamma_{\text{aux}} = \mathsf{PBVI\_step}(\mathcal{M}, \Gamma^{(k)}, \mathcal{B}^{(k+1)})$
 8:     $\mathcal{G}^{(k+1)} = \mathcal{G}^{(k)}$
 9:     for $\boldsymbol{\alpha}_m \in \Gamma_{\text{aux}}$ do
10:         Add node $m$ to $\mathcal{G}^{(k+1)}$
11:         if $\boldsymbol{\alpha}_m \leq \boldsymbol{\alpha}_\ell$ for $\boldsymbol{\alpha}_\ell \in \Gamma^{(k)}$ then
12:             Remove node $\ell$
13:         end if
14:     end for
15:     $\mathcal{G}^{(k+1)} = \mathsf{remove\_unlinked}(\mathcal{G}^{(k+1)})$
16:     $k \leftarrow k + 1$
17: until $\pi^{(k-1)} = \pi^{(k)}$ or $k > K_{\max}$
18: return $\pi^{(k)}$
```

Example 6.12 To illustrate the performance of PBPI, we applied this algorithm to the familiar tiger problem. The algorithm returns the policy depicted in Fig. 6.25(b). The policy can easily be interpreted by inspecting the policy graph: after listening two consecutive steps the tiger behind one of the doors, open that door. The corresponding cost-to-go function and policy are compared with those obtained after 20 iterations of exact VI in Fig. 6.25(a), and we can observe that the two policies coincide.

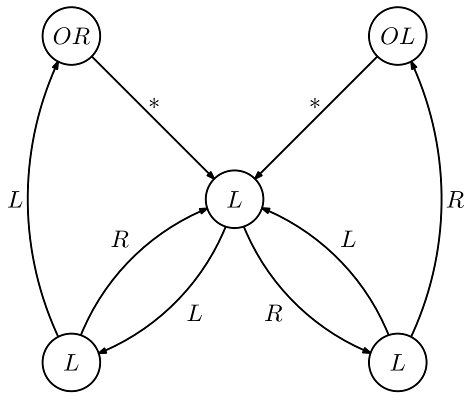
Additionally, as was observed with PERSEUS, the cost-to-go function computed by PBPI is also indistinguishable from the one obtained with exact VI. PBPI completed in 6 iterations that took a total of 226 ms to complete. The resulting cost-to-go function is identical to the one computed by PERSEUS, requiring only 5 α-vectors.

Example 6.13 We applied PBPI to the fault detection example from Section 6.4.3, where we set the number of modules to $N = 4$. In other words, we have a total of 256 states and 4 actions, corresponding to inspecting (I), manufacturing (M), repair (R_1) and replace (R_2).

As with PERSEUS, we varied the number of sampled beliefs and observed how the computation time and the number of nodes in the policy graph varied. The results are depicted in Fig. 6.26, and are qualitatively similar to those observed in PERSEUS. This is not unexpected, as both methods include a



(a) Comparison between the cost-to-go function and policy computed using PBPI and the policy obtained after 20 iterations of VI.



(b) Policy graph obtained by PBPI.

Figure 6.25 Result of the application of PBPI to the tiger problem. The algorithm terminated after 6 iterations that took a total of 226 ms.

PBVI update at its core. Using $5,000$ belief points, PBPI finished after $9,399$ iterations, taking a total of 570 seconds to complete. The resulting policy had a total of 34 nodes and is represented in Fig. 6.27, where larger and lighter nodes have larger in-degree. As with Example 6.11, a detailed interpretation of the resulting policy is difficult. We note, however, that the agent decides to manufacture in most situations, and never to inspect.

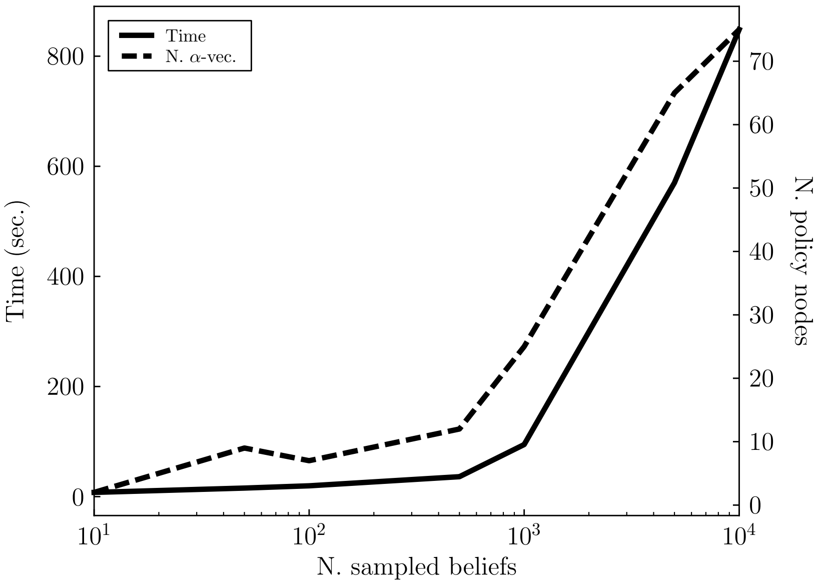


Figure 6.26 Time and number of nodes in the policy graph resulting from applying PBPI with a different number of sampled beliefs.

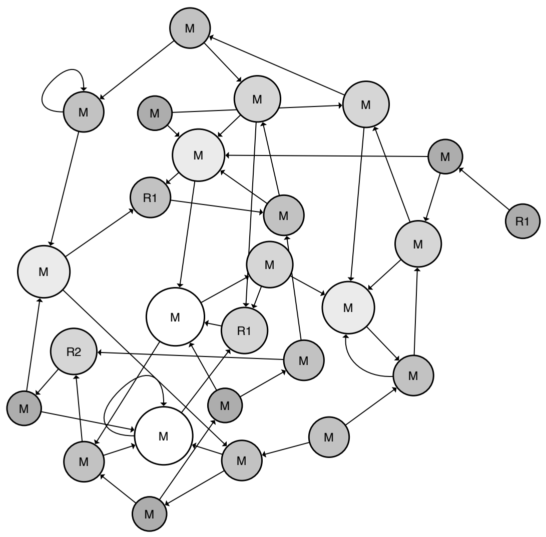


Figure 6.27 Policy graph for the fault detection problem.

6.5.3 Other approaches (⋆)

We conclude this section by providing an overview of other approaches to solve POMDPs in the literature. We discuss only the main ideas, leaving out most of the technical details.

Inference-based planning

Toussaint et al (Toussaint, Harmeling, and Storkey, 2006; Toussaint and Storkey, 2006) proposed an approach for solving MDPs and POMDPs that converts the problem of computing the optimal policy—i.e., the policy that minimizes the discounted cost-to-go incurred by the decision-maker—into an equivalent problem of computing the maximum likelihood parameter from incomplete data.

To explain the main idea behind the proposed approach, let us first consider the fully observable case. Given an MDP $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \{\boldsymbol{P}_a\}, c, \gamma, \mu_0)$, let π denote an arbitrary stationary policy. Suppose that we "sample" the Markov chain induced by policy π at a random time step t, where

$$\mathbb{P}[\mathrm{t} = t] = (1-\gamma)\gamma^t.$$

Let ζ be a binary r.v. such that

$$\mathbb{P}_\pi[\zeta = 1] = c(\mathrm{x}_\mathrm{t}, \mathrm{a}_\mathrm{t}).$$

Note, in particular, that the probabilities governing ζ depend on the time step at which we sample the Markov chain induced by π and the trajectories of the chain. We can bring out the dependence of ζ on π by expanding the probability on the left hand side, to get

$$\begin{aligned}
\mathbb{P}_\pi[\zeta = 1] &= \sum_{t=0^\infty} \mathbb{P}_\pi[\zeta = 1 \mid \mathrm{t} = t]\, \mathbb{P}[\mathrm{t} = t] \\
&= (1-\gamma)\sum_{t=0}^{\infty} \gamma^t \mathbb{P}_\pi[\zeta = 1 \mid \mathrm{t} = t] \\
&= (1-\gamma)\sum_{t=0}^{\infty} \gamma^t \sum_{x_0 \in \mathcal{X}} \mathbb{P}_\pi[\zeta = 1 \mid \mathrm{t} = t, \mathrm{x}_0 = x_0]\, \mathbb{P}_\pi[\mathrm{x}_0 = x_0, \mathrm{t} = t].
\end{aligned}$$

Replacing the definitions, some manipulations yield

$$\begin{aligned}
\mathbb{P}_\pi[\zeta = 1] &= (1-\gamma)\sum_{t=0}^{\infty} \gamma^t \sum_{x_0 \in \mathcal{X}} \mathbb{P}_\pi[\zeta = 1 \mid \mathrm{t} = t, \mathrm{x}_0 = x_0]\, \mu_0(x_0) \\
&= (1-\gamma)\sum_{t=0}^{\infty} \gamma^t \sum_{x_0 \in \mathcal{X}} \mathbb{E}_\pi[\zeta \mid \mathrm{t} = t, \mathrm{x}_0 = x_0]\, \mu_0(x_0) \\
&= (1-\gamma)\sum_{t=0}^{\infty} \gamma^t \sum_{x_0 \in \mathcal{X}} \mathbb{E}_\pi[\mathrm{c}_t \mid \mathrm{x}_0 = x_0]\, \mu_0(x_0) \\
&= (1-\gamma)\sum_{x_0 \in \mathcal{X}} J^\pi(x_0)\mu(x_0).
\end{aligned}$$

Conversely,

$$\mathbb{P}_\pi[\zeta = 0] = 1 - (1-\gamma)\sum_{x_0 \in \mathcal{X}} J^\pi(x_0)\mu(x_0).$$

Therefore, finding the policy π that maximizes $J^\pi(x)$ for all states is equivalent to finding the policy π that maximizes the likelihood $\mathbb{P}_\pi[\zeta = 0]$. This can be done

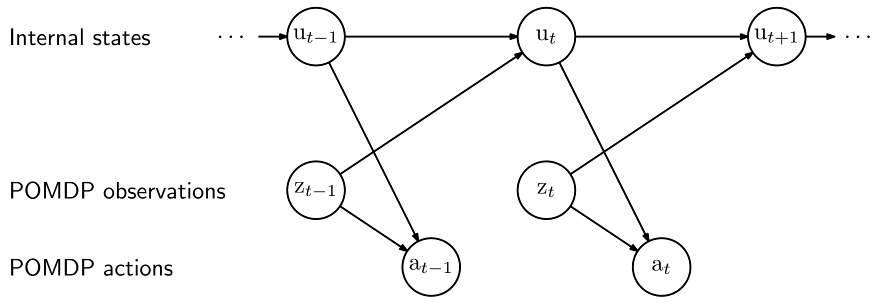


Figure 6.28 Dynamic Bayesian network illustrating the dependence of the actions on the internal controller. At every step t the action a_t is selected depending on the internal state of the controller, $\boldsymbol{b}_t$, and on the current observation, z_t. The controller then transitions to a new internal state, $\boldsymbol{b}_{t+1}$, that depends on $\boldsymbol{b}_t$ and z_t.

VIA the EM algorithm (see Chapter 3), since we want to estimate the maximum likelihood parameter (the policy π) from incomplete data (the r.v. ζ). For general MDPs, Toussaint and Storkey (2006) show that such inference-based approach to computing π^* is similar to policy iteration: the E-step corresponds to a policy evaluation step, where $Q^\pi(x, a)$ is computed for the current policy and the M-step corresponds to a policy improvement step, where a new policy, π_g^Q, is computed from Q^π.

The POMDP version relies on a finite policy representation that is a generalization of the finite state controllers described in Sections 6.4.2 and 6.5.2. In particular, the policy is represented as a pair $(\mathcal{M}_I, \delta_I)$, where $\mathcal{M}_I = (\mathcal{U}_I, \mathcal{Z}, \boldsymbol{P}_I, \mu_I)$ is a controlled Markov chain with $\mathcal{U}_I$ a finite set of internal states, $\boldsymbol{P}_I$ the observation-dependent transition probabilities, and μ_I the initial distribution over the states in $\mathcal{U}_I$. The mapping $\delta_I : \mathcal{U}_I \times \mathcal{Z} \to \Delta(\mathcal{A})$ is a decision rule that maps each element $(u, z) \in \mathcal{U}_I \times \mathcal{Z}$ to a distribution over actions. We write $\delta_I(a \mid u, z)$ to denote the probability of action a after observing z when the controller is in state u.

We again sample the process at a random time step t, where

$$\mathbb{P}[\mathrm{t} = t] = (1 - \gamma)\gamma^t,$$

and define a binary r.v. ζ such that

$$\mathbb{P}_\pi[\zeta = 1] = c(\mathrm{x_t}, \mathrm{a_t}).$$

The likelihood is defined as above, but now depends on all the parameters of the policy, namely:

- The probabilities $\mu_I(u), u \in \mathcal{U}$, governing the initial state of the controller;
- The probabilities $\boldsymbol{P}(u' \mid u, z)$, governing the state transitions of the controller;
- The probabilities $\delta(a \mid u, z)$, governing the action selection.

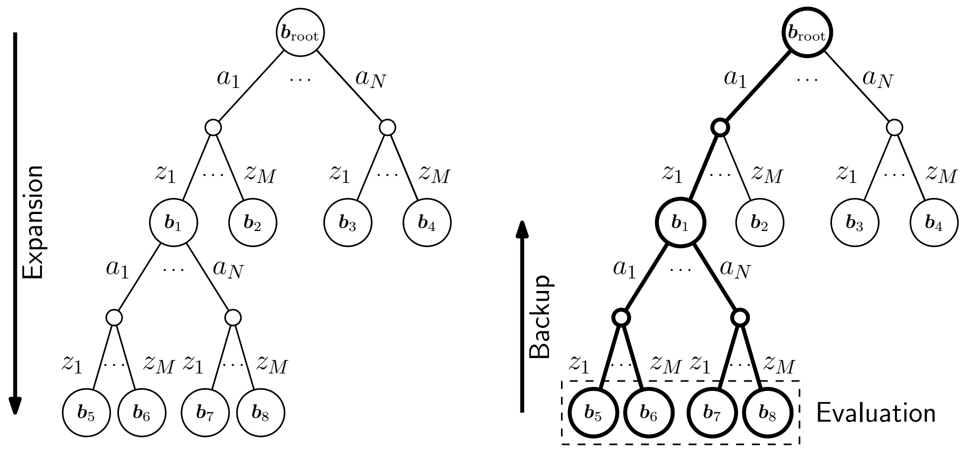


Figure 6.29 Policy computation step in online planning. At each step t, a "belief tree" is built with the agent's belief, $\boldsymbol{b}_t$, as the root node. The tree is built from the root down by considering possible sequences of actions and observations up to a prescribed depth d, and the leaf belief nodes are evaluated using any preferred heuristic. Finally, the values of the leaf nodes are then backed up to the root node, informing the action selection.

The problem can again be solved using the EM algorithm. The resulting algorithm, as argued by Toussaint, Harmeling, and Storkey (2006), iteratively computes a locally optimal finite state controller for the POMDP, and is similar in flavor to the approach of Meuleau et al. (1999). An alternative solution method, proposed by Hoffman et al. (2007), relies on a similar representation but instead uses a variant of Markov chain Monte Carlo (see Chapter 2) with some argued efficiency advantages.

Online planning

The solution methods so far take as input a model $(\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c, \gamma)$ and return the optimal policy for the POMDP (or an approximation thereof). The computed policy can then be adopted by the agent, prescribing at each step t the action that the agent should take as a function of its current belief, $\mathbf{b}_t$. In all these methods, *policy computation* is separated from *policy execution*: the policy is available to the agent only upon completion of the planning algorithm.

In contrast, *online planning approaches* alternate steps of policy computation with steps of policy execution. At each step t, online approaches build an estimate the optimal policy at the current belief, $\boldsymbol{b}_t$. Such estimate is then used to select the action a_t, and the process repeats.

Since policy computation takes place between two decision epochs, it is naturally constrained in the amount of time available for computation. Typical approaches build, at each step t, a "belief tree" with the agent's current belief, $\boldsymbol{b}_t$, as the root node. The tree is expanded to a depth that is either pre-specified or limited by the allotted time budget for the overall computation. Once the tree is constructed, the leaf nodes are evaluated using, for example, the MDP heuristics surveyed in Section 6.5.1. Another alternative is to use an estimate of the cost-to-go function computed using, for example, PBVI. These estimates are then propagated back to the root node using the standard recursion in (6.13).

The quality of the policy computed by online planning algorithms depends critically on two factors:

- *The creation of the belief tree.* Deeper trees will more accurately reflect the long-term impact of the different actions. However, the size of the tree grows exponentially with its depth. Given the time constraints resulting from the online nature of such algorithms, it is imperative that the expansion of the tree favors sequences of actions and observations that are closer to those observed when following the optimal policy.

- The estimation of the cost-to-go associated with each action. The action selection results directly from such estimates. Inaccurate estimates may thus imply that the agent will select its actions in a suboptimal fashion.

Several online planning algorithms exist that differ on their approach to the two above factors. For example, Paquet, Tobin, and Chaib-draa (2005) propose a branch-and-bound approach that keeps upper and lower bounds on the estimated cost-to-go associated to each action. These bounds can then be used to prevent the nodes corresponding to suboptimal actions from being expanded. In another approach, McAllester and Singh (1999) propose a sampling-based approach that avoids expanding all observation nodes. Instead, observations are sampled according to their probabilities and the tree is expanded only more frequent action-observation trajectories. Alternatively, it is possible to expand the belief tree using different heuristics (Ross and Chaib-draa, 2007).

Regarding the estimation of the cost-to-go associated with action, the most recent approaches rely on sampling: to estimate the cost-to-go associated with a given leaf node, a number of *rollouts* are performed using some pre-defined policy. The average (discounted) cost incurred in such rollouts provides an estimate of the expected discounted cost-to-go of the desired leaf note. Rollout-based evaluation was originally proposed by Bertsekas and Castañon (1999) and is behind popular *Monte Carlo tree search* (MCTS) planning methods. An example of MCTS planning for POMDPs can be found in the work of Silver and Veness (2010). We will discuss MCTS at length in Chapter 9.

For additional discussion on online planning for POMDPs, we refer to the survey of Ross, Pineau, et al. (2008).

6.6 Complexity

We conclude this chapter with an overview of the main complexity results concerning POMDPs, for which we start with some nomenclature.

Recall from Chapter 5 that complexity theory is concerned with the "difficulty" of decision problems—i.e., problems that can be formulated as "yes/no" questions. In particular, the complexity of a problem $\mathcal{D}$ is the amount of resources needed, in the worst case, to solve an instance D of $\mathcal{D}$ as a function of the size of D. The size of an instance $D \in \mathcal{D}$, $|D|$, is the number of bits needed to represent D. Resources of interest are usually *time* and *space*.

In Chapter 5 we introduced the class of problems that can be *solved* in polynomial time (denoted as **P**) and the class of problems whose solutions can be *verified* in polynomial time (denoted as **NP**). We also mentioned that problems in **P** are usually considered "easy", while problems in **NP** are usually considered "hard".

A problem $\mathcal{D}$ is in class **PSPACE** if any instance $D \in \mathcal{D}$ admits a solution that can be represented using an amount of space that is polynomial in $|D|$. A solution requiring a polynomial amount of space to represent necessarily takes a polynomial amount of time to be processed (and verified), so $\mathbf{NP} \subset \mathbf{PSPACE}$.[8] Therefore, problems in **PSPACE** are also considered "hard". Finally, a problem $\mathcal{D}$ is *undecidable* when there is no algorithm that yields the correct answer for all instances $D \in \mathcal{D}$.

If $\mathcal{D}$ is an optimization problem, for each instance $D \in \mathcal{D}$, $F(D)$ is the set of *feasible solutions* for D, and $c_D(x)$ is the *cost* of solution $x \in F(D)$. The optimal cost is defined as $c^*_D = \min_{x \in F(D)} c_D(x)$. If (A) is an algorithm that, for each instance $D \in \mathcal{D}$, returns a feasible solution $\mathsf{A}(D) \in F(D)$, we say that A is an *ε-approximation algorithm* if, for any instance $D \in \mathcal{D}$,

$$|c_D(\mathsf{A}(D)) - c^*_D| \leq \varepsilon c_D(\mathsf{A}(D)).$$

Given $\varepsilon > 0$, an optimization problem $\mathcal{D}$ is ε-approximable if there is an algorithm A that, for any $D \in \mathcal{D}$, outputs an ε-approximation in an amount of time that is polynomial in $|D|$.

Like MDPs, POMDPs are not "yes/no" problems. Their complexity is usually formulated in terms of the *policy existence problem*: "given a POMDP and a threshold K, is there a policy that attains a value of K?"[9] Related problems can similarly be expressed as decision problems. For example, the complexity associated with the computation of the best memoryless policy for a POMDP can be analyzed in terms of the problem: "given a POMDP and a threshold K, is there a memoryless policy that attains a value of K?"

We are now in position to discuss the existing complexity results for POMDPs. Initial results on the complexity of POMDPs are due to Papadimitriou and Tsitsiklis (1987), who established that finite-horizon POMDPs are **PSPACE**-complete. Mundhenk et al. (2000) further showed that finite-horizon POMDPs are in **NP** even if we consider only memoryless policies. Finally, Lusena, Goldsmith, and Mundhenk (2001) showed that, for any $\varepsilon > 0$, an ε-approximation for POMDPs does not exist unless if $\mathbf{P} = \mathbf{NP}$ (a statement widely believed to be false).

[8] It is unknown whether the set containment is strict.

[9] The notion of "value" depends on the optimality criterion considered for the POMDP. For example, for finite-horizon POMDPs, the "value" usually corresponds to the total accumulated cost; in infinite-horizon discounted POMDPs, the "value" corresponds to the discounted cost-to-go.

Given the aforementioned complexity results, the class of POMDPs discussed in this chapter—infinite horizon POMDPs with the discounted cost-to-go optimality criterion—is even less likely to hold efficient solutions. Madani, Hanks, and Condon (2003) showed that this class of POMDPs is, in fact, *undecidable*. In that same work, the authors also show that computing the best finite state controller for this class of POMDPs is also undecidable. Lusena, Goldsmith, and Mundhenk (2001) showed that finding the best memoryless policy for any such POMDP is an **NP** problem and that ε-approximations for POMDPs are unlikely to exist. All such results are surveyed and further discussed in the article of Mundhenk (2000).

In light of all such disheartening results, it is perhaps surprising that many of the methods discussed in this chapter—particularly those surveyed in Section 6.5—yield near-optimal solutions in many practical problems. Driven by such observation, several researchers suggested that the analysis conducted within complexity theory is not the best to translate the difficulty of POMDP planning. For example, Hsu, Lee, and Rong (2007) showed that the set of reachable beliefs in POMDPs usually lie in small-dimensional manifold, and that the complexity of planning in a POMDP may be better captured by how well such manifold can be identified. They propose *the covering number* as a criterion to measure the difficulty of a given POMDP problem.

Given a POMDP $(\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c, \gamma)$ with initial belief $\boldsymbol{b}_0$, let $\mathcal{B}_{\text{reach}} \subset \mathcal{B}$ denote the set of beliefs reachable from $\boldsymbol{b}_0$. Let $C(\delta)$ denote the δ-covering number of $\mathcal{B}_{\text{reach}}$, i.e., the smallest number of balls of radius δ needed to cover $\mathcal{B}_{\text{reach}}$. The key result established in the work of Hsu, Lee, and Rong (2007) is that an ε-approximation for $J^*(b_0)$ can be constructed in an amount of time that is polynomial in $C(k\varepsilon(1-\gamma)^2/\gamma)$, for some constant k. In order to do so, however, it is necessary to know such cover, a task that is hard for many POMDPs. Nevertheless, these result explain, at least to some extent, the large success behind point-based methods, which focus their computational effort on the set of reachable beliefs. Some posterior point-based methods—such as SARSOP, for example (Kurniawati, Hsu, and Lee, 2008)—have been designed to approximately estimate the cover of $\mathcal{B}_{\text{reach}}$. Z. Zhang, Littman, and Chen (2012) further explore the properties of the covering number, both theoretically—by showing that the covering number may indeed be the best criterion by which to evaluate the difficulty of a POMDP—and in practice, proposing an algorithm that seeks to exploit the structure of the reachable belief space to the most.

6.7 Bibliographical notes

POMDPs are among the richest models for single-agent decision problems, and have been applied in a vast range of domains. The examples described in this chapter are merely illustrative, and adapted from several early works from the POMDP literature. For example, the tiger example is a classical POMDP problem introduced in the work of Cassandra, Kaelbling, and Littman (1994). It is a very didactic problem that allows an easy visualization of many of the challenges posed by POMDPs. The slotted ALOHA and fault detection problems were adapted from the thesis of Cassandra (1998). The shuttle problem is adapted from the work of Chrisman (1992). A large repository of POMDP benchmark problems—that includes those used in this chapter—is available at `http://www.pomdp.org/`.

Many more applications can be found in the literature. For example, Ellis, Jiang, and Corotis (1995) propose the use of a POMDP for planning infrastructure inspection and maintenance activities. Hauskrecht and Fraser (2000) used POMDPs to plan the treatment of patients with ischemic heart disease and Boger et al. (2005) describe the application of POMDPs to the assistance of people with dementia. In another work, Ong et al. (2010) apply POMDPs in AUV navigation. Kochenderfer (2015) describes the use of POMDPs in aircraft collision avoidance. Young et al. (2013) surveys the use of POMDPs in dialogue systems. As a last example, Chadès et al. (2008) describe the use of a POMDP to decide when to survey (rather than manage) populations of endangered species.

◇

Research into POMDPs has a long history, originally in the operations research community and only more recently in the AI community. Most fundamental results—such as the equivalence between POMDPs and the corresponding belief-MDPs (Åström, 1965) or the piecewise linearity and concavity of the cost-to-go function (Sondik, 1971)—date back to the early days of POMDP research. The first solution methods for POMDPs are variations of value-iteration and due to Sondik (1971), Monahan (1982), and Cheng (1988). These early works set most of the fundamental ideas upon which posterior methods were built.

The two value-iteration algorithms in Section 6.4 are currently the most efficient exact solution methods. The witness algorithm is due to Littman (1994); a detailed account can be found in the article of Kaelbling, Littman, and Cassandra (1998). Incremental pruning was originally proposed by N. Zhang and W. Liu (1996) and further established in the work of Cassandra, Littman, and N. Zhang (1997).

Sondik (1978) proposed the first policy iteration algorithm for POMDPs. In it, Sondik proposes the first finite representation for POMDP policies as polyhedral divisions of the belief space. This representation was, however, rather involved and impractical from a computational perspective. The version discussed in Section 6.4 is due to Hansen (1997, 1998).

◇

The MLS and AV heuristics discussed in Section 6.5.1 were proposed by Cassandra, Kaelbling, and Kurien (1996). Q-MDP was proposed by Littman, Cassandra, and Kaelbling (1995), while FIB was proposed by Hauskrecht (2000). Of all the MDP heuristics discussed in this chapter, the best known is, perhaps, Q-MDP, both due to its intuitive interpretation and its ease of computation. Several works report reasonable performance using Q-MDP in "well-behaved" domains, such as indoor robotic navigation (Cassandra, 1998). Q-MDP also provides a natural upper bound of J^* that is used, for example, in branch-and-bound approaches such as those discussed in Section 6.5.3. FIB, while providing sounder approximations for J^*, is slightly more complex to compute and less intuitive to interpret. A detailed discussion of the merits of MDP heuristics (including FIB and ither variations) can be found in the original work of Hauskrecht (2000).

More recently, some works addressed the propensity of Q-MDP not to take information gathering actions. For example, Cassandra (1998) proposes a *dual-mode control*, which alternates between cost avoidance actions and information gathering actions based on the amount of uncertainty in the belief space. In a similar line of work, F. Melo and Ribeiro (2006) propose a heuristic based on the concept of *transition entropy*, which measures how much entropy in the belief a given transition induces. The heuristic weights together cost and transition entropy minimization to output an estimate of the cost-to-go function.

◇

The fundamental idea behind point-based methods—that of using a finite set of points to render value iteration more efficient—is, perhaps, due to Lovejoy (1991), who proposed the use of a uniform grid of points. The current work on PBVI was pioneered by Pineau, Gordon, and Thrun (2003). The PERSEUS algorithm discussed in this chapter is due to Spaan and Vlassis (2005) and remains one of the most widely used approaches to POMDPs. For example, it has been combined with efficient POMDP representations (for example, algebraic decision diagrams) to solve POMDPs with close to 50 million states (Boger et al., 2005; Poupart, 2005). Nevertheless, two key steps in PERSEUS depend on random sampling, which may negatively impact the performance of the method in very large-scale problems. The first step that depends on random sampling is the construction of the set $\mathcal{B}_{\text{sample}}$, which relies on undirected exploration of the belief space. Ensuring that the set $\mathcal{B}_{\text{sample}}$ provides a comprehensive coverage of the reachable belief space is a challenging problem, and several alternative approaches construct the belief set $\mathcal{B}_{\text{sample}}$ to maximize such coverage—SARSOP and PGVI, for example (Kurniawati, Hsu, and Lee, 2008; Z. Zhang, Hsu, and Lee, 2014).

The second critical step of PERSEUS that may suffer from random sampling is the order by which belief points in $\mathcal{B}_{\text{sample}}$ are updated. To address such difficulty, *heuristic search value iteration* (Smith and Simmons, 2005) maintains upper and lower bounds on the optimal cost-to-go function, which are used to select belief points where uncertainty in J is larger, and updates are performed using the inverse order by which belief points are visited. Other variants that exploit the same idea include forward-search value iteration (Shani, Brafman, and Shimony, 2007), SARSOP (Kurniawati, Hsu, and Lee, 2008) and GapMin (Poupart, K. Kim, and D. Kim, 2011). We note that this class of point-based methods—which maintain both upper and lower bounds on J^*—provide another use for MDP heuristics such as Q-MDP and FBI, as they yield natural upper bounds for J^*. For an comprehensive overview and extensive discussion of point-based value iteration, we refer to the survey of Shani, Pineau, and Kaplow (2013).

Finally, the idea of bounding the representation of a policy is also old—already present, for example, in the work of McCallum (1992) on utile distinction memory. However, it is perhaps in the work of Meuleau et al., 1999 that is first clearly formulated. Meuleau et al. (1999) show that finding the best finite state controller for a POMDP is **NP**-hard, and propose a gradient ascent approach to address such computation. The use of finite-state controllers in a context of policy iteration was explored in the work of Poupart and Boutilier (2003, 2004). In their work they propose to forcefully bound the size of the policy graph representation, while allowing for *stochastic controllers* that can compensate for the bounded controller size. The PBPI algorithm discussed in Section 6.5.2 is due to Ji et al. (2007).

6.8 Proofs

Proof of Proposition 6.1

By virtue of the fact that $\mathbf{b}_t$ is a sufficient statistic for h_t,

$$\begin{aligned}
&\mathbb{P}\left[\mathbf{b}_{t+1} = \boldsymbol{b}' \mid \mathrm{a}_t = a, \mathrm{h}_t = h\right] \\
&\quad= \sum_{z\in\mathcal{Z}} \mathbb{P}\left[\mathbf{b}_{t+1} = \boldsymbol{b}' \mid \mathrm{z}_{t+1} = z, \mathrm{a}_t = a, \mathrm{h}_t = h\right] \mathbb{P}\left[\mathrm{z}_{t+1} = z \mid \mathrm{a}_t = a, \mathrm{h}_t = h\right] \\
&\quad= \sum_{z\in\mathcal{Z}} \mathbb{P}\left[\mathbf{b}_{t+1} = \boldsymbol{b}' \mid \mathrm{z}_{t+1} = z, \mathrm{a}_t = a, \mathbf{b}_t = \boldsymbol{b}\right] \mathbb{P}\left[\mathrm{z}_{t+1} = z \mid \mathrm{a}_t = a, \mathrm{h}_t = h\right],
\end{aligned}$$

where $\boldsymbol{b}$ is the belief corresponding to the history h and the last equality follows from (6.6). Moreover, considering only the second term in the summation, we have that

$$\begin{aligned}
&\mathbb{P}\left[\mathrm{z}_{t+1} = z \mid \mathrm{a}_t = a, \mathrm{h}_t = h\right] \\
&= \sum_{y\in\mathcal{X}} \mathbb{P}\left[\mathrm{z}_{t+1} = z \mid \mathrm{x}_{t+1} = y, \mathrm{a}_t = a, \mathrm{h}_t = h\right] \mathbb{P}\left[\mathrm{x}_{t+1} = y \mid \mathrm{a}_t = a, \mathrm{h}_t = h\right] \\
&= \sum_{x,y\in\mathcal{X}} \boldsymbol{O}(z \mid y, a)\mathbb{P}\left[\mathrm{x}_{t+1} = y \mid \mathrm{x}_t = x, \mathrm{a}_t = a, \mathrm{h}_t = h\right] \mathbb{P}\left[\mathrm{x}_t = x \mid \mathrm{a}_t = a, \mathrm{h}_t = h\right] \\
&= \sum_{x,y\in\mathcal{X}} \boldsymbol{O}(z \mid y, a)\boldsymbol{P}(y \mid x, a)\mathbb{P}\left[\mathrm{x}_t = x \mid \mathrm{h}_t = h\right] \\
&= \sum_{x,y\in\mathcal{X}} \boldsymbol{O}(z \mid y, a)\boldsymbol{P}(y \mid x, a)\boldsymbol{b}(x) \\
&= \mathbb{P}\left[\mathrm{z}_{t+1} = z \mid \mathrm{a}_t = a, \mathbf{b}_t = \boldsymbol{b}\right],
\end{aligned}$$

where the third equality follows from the fact that a_t is a function of h_t. The conclusion follows.

Proof of Lemma 6.2

We establish the result by induction on t. The conclusion is immediate for $t = 0$. Suppose then that, for some $t \in \mathbb{N}$, (6.11) holds. We have that

$$\begin{aligned}
&\mathbb{P}_{\hat{\pi}}\left[\mathbf{b}_{t+1} = \boldsymbol{b} \mid \mathbf{b}_{0:t} = \boldsymbol{b}_{0:t}, \mathbf{a}_{0:t} = \boldsymbol{a}_{0:t}\right] \\
&\quad= \mathbb{P}_{\pi}\left[\mathbf{b}_{t+1} = \boldsymbol{b} \mid \mathbf{b}_{0:t} = \boldsymbol{b}_{0:t}, \mathbf{a}_{0:t} = \boldsymbol{a}_{0:t}\right] \\
&\quad= \hat{\boldsymbol{P}}(\boldsymbol{b} \mid \boldsymbol{b}_t, a_t).
\end{aligned} \tag{6.30}$$

Moreover, from the definition of $\hat{\pi}$,

$$\begin{aligned}
&\mathbb{P}_{\hat{\pi}}\left[\mathrm{a}_{t+1} = a \mid \mathbf{b}_{0:t+1} = \boldsymbol{b}_{0:t+1}, \mathbf{a}_{0:t} = \boldsymbol{a}_{0:t}\right] \\
&= \sum_{\boldsymbol{z}_{0:t+1}} \pi(a \mid \boldsymbol{z}_{0:t+1}, \boldsymbol{a}_{0:t}) \mathbb{P}_{\pi}\left[\mathbf{z}_{0:t+1} = \boldsymbol{z}_{0:t+1} \mid \mathbf{b}_{0:t+1} = \boldsymbol{b}_{0:t+1}, \mathbf{a}_{0:t} = \boldsymbol{a}_{0:t}\right] \qquad (6.31) \\
&= \mathbb{P}_{\pi}\left[\mathrm{a}_{t+1} = a \mid \mathbf{b}_{0:t+1} = \boldsymbol{b}_{0:t+1}, \mathbf{a}_{0:t} = \boldsymbol{a}_{0:t}\right].
\end{aligned}$$

Combining (6.30) and (6.31), we immediately have that

$$\begin{aligned}
&\mathbb{P}_{\hat{\pi}}\left[\mathbf{b}_{t+1} = \boldsymbol{b}, \mathbf{a}_{t+1} = a \mid \mathbf{b}_{0:t} = \boldsymbol{b}_{0:t}, \mathbf{a}_{0:t} = \boldsymbol{a}_{0:t}\right] \\
&= \mathbb{P}_{\pi}\left[\mathbf{b}_{t+1} = \boldsymbol{b}, \mathbf{a}_{t+1} = a \mid \mathbf{b}_{0:t} = \boldsymbol{b}_{0:t}, \mathbf{a}_{0:t} = \boldsymbol{a}_{0:t}\right].
\end{aligned}$$

The desired conclusion follows from the induction hypothesis and the total probability law.

Proof of Proposition 6.4

To establish the result, we use the following facts.

Lemma 6.10. *Let $F_1, F_2 : \mathbb{R}^N \to \mathbb{R}$ denote two piecewise linear and concave functions, with*

$$F_1(\boldsymbol{x}) = \min_{\boldsymbol{\alpha} \in \Gamma_1} \left[\alpha_0 + \sum_{n=1}^{N} \alpha_n x_n\right] \qquad \textit{and} \qquad F_2(\boldsymbol{x}) = \min_{\boldsymbol{\beta} \in \Gamma_2} \left[\beta_0 + \sum_{n=1}^{N} \beta_n x_n\right],$$

with $\Gamma_1, \Gamma_2 \subset \mathbb{R}^{N+1}$. Then, the functions $G_1(\boldsymbol{x}) = F_1(\boldsymbol{x}) + F_2(\boldsymbol{x})$ and $G_2(\boldsymbol{x}) = \min\{F_1(\boldsymbol{x}), F_2(\boldsymbol{x})\}$ are also piecewise linear and concave.

Proof. Let $\Gamma = \Gamma_1 \oplus \Gamma_2$, i.e.,

$$\Gamma = \left\{\boldsymbol{\gamma} \in \mathbb{R}^{N+1} \mid \boldsymbol{\gamma} = \boldsymbol{\alpha} + \boldsymbol{\beta}, \boldsymbol{\alpha} \in \Gamma_1, \boldsymbol{\beta} \in \Gamma_2\right\}.$$

Clearly,

$$G_1(\boldsymbol{x}) \overset{\text{def}}{=} F_1(\boldsymbol{x}) + F_2(\boldsymbol{x}) = \min_{\boldsymbol{\gamma} \in \Gamma} \left[\gamma_0 + \sum_{n=1}^{N} \gamma_n x_n\right],$$

and G_1 is PWLC. On the other hand, if we let $\Gamma = \Gamma_1 \cup \Gamma_2$, we have that

$$G_2(\boldsymbol{x}) \overset{\text{def}}{=} \max\{F_1(\boldsymbol{x}), F_2(\boldsymbol{x})\} = \min_{\boldsymbol{\gamma} \in \Gamma} \left[\gamma_0 + \sum_{n=1}^{N} \gamma_n x_n\right],$$

and the conclusion follows. □

The statement of Proposition 6.4 can be established by induction on k. For $k = 0$ the result trivially follows. As induction hypothesis suppose that, for some k,

$$J^{(k)}(\boldsymbol{b}) = \min_{\boldsymbol{\alpha}^{(k)} \in \Gamma^{(k)}} \boldsymbol{b} \cdot \boldsymbol{\alpha}^{(k)}.$$

Let $\boldsymbol{o}_{a,z}$ denote the vector in $\mathbb{R}^{|\mathcal{X}|}$ with xth component

$$\boldsymbol{o}_{a,z}(x) = \boldsymbol{O}(z \mid x, a).$$

Then, in matrix notation, we have that

$$J^{(k+1)}(\boldsymbol{b}) = \min_{a \in \mathcal{A}} \left[\boldsymbol{b}\,\boldsymbol{C}_{:,a} + \gamma \sum_{z \in \mathcal{Z}} \boldsymbol{b}\,\boldsymbol{P}_a\,\boldsymbol{o}_{z,a} \min_{\alpha^{(k)} \in \Gamma^{(k)}} \boldsymbol{B}(\boldsymbol{b}, z, a) \cdot \boldsymbol{\alpha}^{(k)} \right].$$

Writing explicitly the belief update operator $\boldsymbol{B}$ yields

$$\begin{aligned} J^{(k+1)}(\boldsymbol{b}) &= \min_{a \in \mathcal{A}} \left[\boldsymbol{b}\,\boldsymbol{C}_{:,a} + \gamma \sum_{z \in \mathcal{Z}} \boldsymbol{b}\,\boldsymbol{P}_a\,\boldsymbol{o}_{z,a} \min_{\alpha^{(k)} \in \Gamma^{(k)}} \frac{\boldsymbol{b}\,\boldsymbol{P}_a\ \mathrm{diag}(\boldsymbol{o}_{z,a})}{\boldsymbol{b}\,\boldsymbol{P}_a\,\boldsymbol{o}_{z,a}} \cdot \boldsymbol{\alpha}^{(k)} \right] \\ &= \min_{a \in \mathcal{A}} \left[\boldsymbol{b}\,\boldsymbol{C}_{:,a} + \gamma \sum_{z \in \mathcal{Z}} \min_{\alpha^{(k)} \in \Gamma^{(k)}} \boldsymbol{b}\,\boldsymbol{P}_a\ \mathrm{diag}(\boldsymbol{o}_{z,a})\,\boldsymbol{\alpha}^{(k)} \right] \\ &= \min_{a \in \mathcal{A}} \sum_{z \in \mathcal{Z}} \min_{\alpha^{(k)} \in \Gamma^{(k)}} \boldsymbol{b} \cdot \left[\frac{1}{|\mathcal{Z}|} \boldsymbol{C}_{:,a} + \gamma \boldsymbol{P}_a\ \mathrm{diag}(\boldsymbol{o}_{z,a})\,\boldsymbol{\alpha}^{(k)} \right]. \end{aligned} \tag{6.32}$$

The first statement follows from Lemma 6.10, noting that $J^{(k+1)}$ can be obtained by minimizing/adding PWLC functions.

The second statement, in turn, follows from (6.32) and Lemma 6.10, by letting

$$\Gamma_{a,z}^{(k+1)} = \left\{ \frac{1}{|\mathcal{Z}|} \boldsymbol{C}_{:,a} + \gamma \boldsymbol{P}_a\ \mathrm{diag}(\boldsymbol{o}_{z,a})\,\boldsymbol{\alpha}^{(k)}, \boldsymbol{\alpha}^{(k)} \in \Gamma^{(k)} \right\}$$

and

$$\Gamma_a^{(k+1)} = \bigoplus_{z \in \mathcal{Z}} \Gamma_{a,z}.$$

Proof of Proposition 6.5

The algorithm rests on the following result.

Lemma 6.11. *For any α-vector $\boldsymbol{\alpha} \in \Gamma_a^{(k)}$ there is a belief $\boldsymbol{b} \in \mathcal{B}$ such that $\boldsymbol{b} \cdot \boldsymbol{\alpha} > \boldsymbol{b} \cdot \boldsymbol{\alpha}'$, with $\boldsymbol{\alpha}' \in \Gamma_a^{(k)}$, if and only if there is a neighbor $\boldsymbol{v} \in N(\boldsymbol{\alpha})$ such that $\boldsymbol{b} \cdot \boldsymbol{\alpha} > \boldsymbol{b} \cdot \boldsymbol{v}$.*

Proof. The existence of a dominated neighbor trivially implies the existence of a dominated vector. As for the converse implication, suppose that $\boldsymbol{b} \cdot \boldsymbol{\alpha} > \boldsymbol{b} \cdot \boldsymbol{\alpha}'$ for some belief $\boldsymbol{b}$ and some $\boldsymbol{\alpha}' \in \Gamma_a^{(k)}$. Then, we can write

$$\boldsymbol{b} \cdot \sum_{z \in \mathcal{Z}} \boldsymbol{\alpha}_{z,a} > \boldsymbol{b} \cdot \sum_{z \in \mathcal{Z}} \boldsymbol{\alpha}'_{z,a}$$

which implies that there is some $z' \in \mathcal{Z}$ such that $\boldsymbol{b} \cdot \boldsymbol{\alpha}_{a,z'} > \boldsymbol{b} \cdot \boldsymbol{\alpha}'_{a,z'}$. Then, the vector

$$\boldsymbol{v} = \sum_{z \neq z'} \boldsymbol{\alpha}_{z,a} + \boldsymbol{\alpha}'_{z',a}$$

is a neighbor of $\boldsymbol{\alpha}$ and verifies the statement of the lemma. □

The result now follows from the following facts:

- By construction, a vectors $\boldsymbol{\alpha}$ is added to $\Gamma^{(k)}$ only if there is some belief $\boldsymbol{b}$ such that $\boldsymbol{b} \cdot \boldsymbol{\alpha} < \boldsymbol{b} \cdot \boldsymbol{\alpha}'$, for all $\boldsymbol{\alpha}' \in \Gamma^{(k)}$. Therefore, the witness region for such vector is non-empty, and $\boldsymbol{\alpha}$ should indeed be in $\Gamma^{(k)}$.
- No vector is added twice. In fact, if a vector $\boldsymbol{\alpha} \notin \Gamma^{(k)}$ has a witness $\boldsymbol{b}$, then $\boldsymbol{b} \cdot \boldsymbol{\alpha} < \boldsymbol{b} \cdot \boldsymbol{\alpha}'$, for all $\boldsymbol{\alpha}' \in \Gamma^{(k)}$, meaning that $\boldsymbol{\alpha}^*(\boldsymbol{b}) \notin \Gamma^{(k)}$.
- Since the algorithm never adds a vector twice and $\Gamma_a^{(k)}$ is finite, each vector $\boldsymbol{\alpha} \in \Gamma_a^{(k)}$ has only a finite number of neighbors. Therefore, each vector is only added to Q a finite number of times and the cycle between lines 4 and 17 must terminate.
- Finally, suppose that the returned set $\Gamma^{(k)}$ is incomplete. Then, there is a belief $\boldsymbol{b}_0 \in \mathcal{B}$ and a vector $\boldsymbol{\alpha}_0 \in \Gamma_a^{(k)}$ such that

  $$\boldsymbol{b}_0 \cdot \boldsymbol{\alpha}_0 < \boldsymbol{b}_0 \cdot \boldsymbol{\alpha},$$

  for any $\boldsymbol{\alpha} \in \Gamma^{(k)}$, where $\boldsymbol{\alpha} \notin \Gamma^{(k)}$. Let

  $$\hat{\boldsymbol{\alpha}}^* = \operatorname*{argmin}_{\boldsymbol{\alpha} \in \Gamma^{(k)}} \boldsymbol{b} \cdot \boldsymbol{\alpha}.$$

  Then, by Lemma 6.11, there is $\boldsymbol{\alpha}' \in N(\hat{\boldsymbol{\alpha}}^*)$ such that $\boldsymbol{b}_0 \cdot \boldsymbol{\alpha}' < \boldsymbol{b}_0 \cdot \hat{\boldsymbol{\alpha}}^*$. However, since $\boldsymbol{\alpha}'$ was removed from Q, it cannot dominate any vector in $\Gamma^{(k)}$, which is a contradiction. Therefore, $\Gamma^{(k)}$ must be complete.

Proof of Proposition 6.7

Let us first consider the error incurred in one iteration of PBVI. Let J denote some PWLC cost-to-go function, represented by a finite set of α-vectors, and $\mathcal{B}_{\text{sample}} \subset \mathcal{B}$ a finite set of belief points. We write T_{VI} to denote the exact VI operator and T_{PBVI} to denote the PBVI operator, which computes an updated set of α-vectors for only the beliefs in $\mathcal{B}_{\text{sample}}$.

Given an arbitrary belief $\boldsymbol{b} \in \mathcal{B}$, let $\boldsymbol{b}_0$ denote the vector in $\mathcal{B}_{\text{sample}}$ that is closest to $\boldsymbol{b}$, i.e.,

$$\boldsymbol{b}_0 = \operatorname*{argmin}_{\boldsymbol{b}' \in \mathcal{B}_{\text{sample}}} \|\boldsymbol{b} - \boldsymbol{b}'\|_1 .$$

Suppose that we apply T_{VI} to J, yielding a new cost-to-go function $\mathsf{T}_{\text{VI}}J$, and let $\boldsymbol{\alpha}, \boldsymbol{\alpha}_0 \in \Gamma^{(k)}$ be such that $(\mathsf{T}_{\text{VI}}J)(\boldsymbol{b}) = \boldsymbol{\alpha} \cdot \boldsymbol{b}$ and $(\mathsf{T}_{\text{VI}}J)(\boldsymbol{b}_0) = \boldsymbol{\alpha}_0 \cdot \boldsymbol{b}_0$. If we compute $(\mathsf{T}_{\text{PBVI}}J)$—i.e., considering only the α-vectors in $\mathcal{B}_{\text{sample}}$—the error incurred at $\boldsymbol{b}$ verifies

$$|(\mathsf{T}_{\text{VI}}J)(\boldsymbol{b}) - (\mathsf{T}_{\text{PBVI}}J)(\boldsymbol{b})| \leq |(\boldsymbol{\alpha} - \boldsymbol{\alpha}_0) \cdot \boldsymbol{b}|.$$

Since $\boldsymbol{\alpha} \cdot \boldsymbol{b} \leq \boldsymbol{\alpha}_0 \cdot \boldsymbol{b}$, we immediately get that

$$\begin{aligned}|(\mathsf{T}_{\text{VI}}J)(\boldsymbol{b}) - (\mathsf{T}_{\text{PBVI}}J)(\boldsymbol{b})| &\leq \boldsymbol{\alpha}_0 \cdot \boldsymbol{b} - \boldsymbol{\alpha} \cdot \boldsymbol{b} \\ &= \boldsymbol{\alpha}_0 \cdot \boldsymbol{b} - \boldsymbol{\alpha} \cdot \boldsymbol{b} + \boldsymbol{\alpha}_0 \cdot \boldsymbol{b}_0 - \boldsymbol{\alpha}_0 \cdot \boldsymbol{b}_0.\end{aligned}$$

Moreover, since $\boldsymbol{\alpha}_0 \cdot \boldsymbol{b}_0 \leq \boldsymbol{\alpha} \cdot \boldsymbol{b}_0$,

$$\begin{aligned}|(\mathsf{T}_{\text{VI}}J)(\boldsymbol{b}) - (\mathsf{T}_{\text{PBVI}}J)(\boldsymbol{b})| &\leq \boldsymbol{\alpha}_0 \cdot \boldsymbol{b} - \boldsymbol{\alpha} \cdot \boldsymbol{b} + \boldsymbol{\alpha} \cdot \boldsymbol{b}_0 - \boldsymbol{\alpha}_0 \cdot \boldsymbol{b}_0 \\ &\leq (\boldsymbol{\alpha}_0 - \boldsymbol{\alpha}) \cdot (\boldsymbol{b} - \boldsymbol{b}_0) \\ &\leq \|\boldsymbol{\alpha}_0 - \boldsymbol{\alpha}\|_\infty \|\boldsymbol{b} - \boldsymbol{b}_0\|_1 \\ &\leq \frac{\varepsilon(\mathcal{B}_{\text{sample}})}{1-\gamma}.\end{aligned}$$

Since $\boldsymbol{b}$ is arbitrary, it follows that

$$\|\mathsf{T}_{\text{VI}}J - \mathsf{T}_{\text{PBVI}}J\|_\infty \leq \frac{\varepsilon(\mathcal{B}_{\text{sample}})}{1-\gamma}. \tag{6.33}$$

We can now apply the bound in (6.33) recursively to get

$$\begin{aligned}\left\|J_{\text{VI}}^{(k+1)} - J_{\text{PBVI}}^{(k+1)}\right\|_\infty &= \left\|\mathsf{T}_{\text{VI}}J_{\text{VI}}^{(k)} - \mathsf{T}_{\text{PBVI}}J_{\text{PBVI}}^{(k)}\right\|_\infty \\ &\leq \left\|\mathsf{T}_{\text{VI}}J_{\text{VI}}^{(k)} - \mathsf{T}_{\text{VI}}J_{\text{PBVI}}^{(k)}\right\|_\infty + \left\|\mathsf{T}_{\text{VI}}J_{\text{PBVI}}^{(k)} - \mathsf{T}_{\text{PBVI}}J_{\text{PBVI}}^{(k)}\right\|_\infty \\ &\leq \gamma\left\|J_{\text{VI}}^{(k)} - J_{\text{PBVI}}^{(k)}\right\|_\infty + \frac{\varepsilon(\mathcal{B}_{\text{sample}})}{1-\gamma} \qquad \cdots \\ &\leq \sum_{\kappa=0}^{k} \gamma^\kappa \frac{\varepsilon(\mathcal{B}_{\text{sample}})}{1-\gamma} + \gamma^{k+1}\left\|J_{\text{VI}}^{(0)} - J_{\text{PBVI}}^{(0)}\right\|_\infty.\end{aligned}$$

Since $J_{\text{VI}}^{(0)} = J_{\text{PBVI}}^{(0)} \equiv 0$, the conclusion follows.

6.9 Exercises

Exercise 6.1.

Let $\mathcal{M}$ be an arbitrary POMDP $(\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c, \gamma, \boldsymbol{b}_0)$ and $\hat{\mathcal{M}}$ the corresponding belief-MDP. Let $\pi : \mathcal{H} \to \mathcal{A}$ denote an arbitrary POMDP policy, and $\hat{\pi}$ the belief-MDP policy defined as

$$\hat{\pi}_t(a \mid \boldsymbol{b}) = \sum_{h_t \in \mathcal{H}_t} \pi(a \mid h_t)\mathbb{P}_\pi\left[\mathrm{h}_t = h_t \mid \mathbf{b}_t = \boldsymbol{b}\right].$$

Show that π and $\hat{\pi}$ induce the same distribution over histories/beliefs, i.e., given

$$\mathbb{P}_\pi\left[\mathrm{h}_t = h, \mathrm{a}_t = a \mid \mathrm{x}_0 \sim \boldsymbol{b}_0\right] = \mathbb{P}_{\hat{\pi}}\left[\mathrm{h}_t = h, \mathrm{a}_t = a \mid \mathrm{x}_0 \sim \boldsymbol{b}_0\right],$$

for any $t \in \mathbb{N}$.

Exercise 6.2.
Prove Proposition 6.6.

Exercise 6.3.
Given a POMDP $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \mathcal{Z}, \{\boldsymbol{P}_a\}, \{\boldsymbol{O}_a\}, c, \gamma)$, show that

$$\|J^*(\boldsymbol{b}_1) - J^*(\boldsymbol{b}_2)\| \leq \frac{1}{1-\gamma} \|\boldsymbol{b}_1 - \boldsymbol{b}_2\|,$$

for any $\boldsymbol{b}_1, \boldsymbol{b}_2 \in \mathcal{B}$.

Suggestion: Use the fact that J^* admits an arbitrarily close approximation that is PWLC.

Exercise 6.4.
Prove Corollary 6.8.

Exercise 6.5.
Prove Proposition 6.9.