15 *Partially Observable Markov Decision Processes*

15.1 Motivation

This chapter discusses algorithms for the partially observable robot control problem. These algorithms address both the uncertainty in measurement and the uncertainty in control effects. They generalize the value iteration algorithm discussed in the previous chapter, which was restricted to action effect uncertainty. The framework studied here is known as *partially observable Markov decision processes*, or *POMDPs*. This name was coined in the operations research literature. The term *partial* indicates that the state of the world cannot be sensed directly. Instead, the measurements received by the robot are incomplete and usually noisy projections of this state.

As has been discussed in so many chapters of this book, partial observability implies that the robot has to estimate a posterior distribution over possible world states. Algorithms for finding the optimal control policy exist for finite worlds, where the state space, the action space, the space of observations, and the planning horizon T are all finite. Unfortunately, these exact methods are computationally involved. For the more interesting continuous case, the best known algorithms are approximate.

All algorithms studied in this chapter build on the value iteration approach discussed previously. Let us restate Equation (14.14), which is the central update equation in value iteration in MDPs:

$$V_T(x) \quad = \quad \gamma \max_u \left[r(x,u) + \int V_{T-1}(x')\, p(x' \mid u, x)\, dx' \right] \tag{15.1}$$

with $V_1(x) \;=\; \gamma \max_u \; r(x,u)$. In POMDPs, we apply the very same idea. However, the state x it not observable. The robot has to make its decision in the belief state, which is the space of posterior distributions over states.

Throughout this and the next chapters, we will abbreviate a belief by the symbol b, instead of the more elaborate bel used in previous chapters.

POMDPs compute a value function over belief space:

$$V_T(b) \quad = \quad \gamma \max_u \left[r(b,u) + \int V_{T-1}(b')\, p(b' \mid u, b)\, db' \right] \tag{15.2}$$

with $V_1(b) \ = \ \gamma \max_u E_x[(x,u)]$. The induced control policy is as follows:

$$\pi_T(b) \quad = \quad \operatorname*{argmax}_u \left[r(b,u) + \int V_{T-1}(b')\, p(b' \mid u, b)\, db' \right] \tag{15.3}$$

A belief is a probability distribution; thus, each value in a POMDP is a function of an entire probability distribution. This is problematic. If the state space is finite, the belief space is continuous, since it is the space of all distributions over the state space. Thus, there is a continuum of different values; whereas there was only a finite number of different values in the MDP case. The situation is even more delicate for continuous state spaces, where the belief space is an infinitely-dimensional continuum.

An additional complication arises from the computational properties of the value function calculation. Equations (15.2) and (15.3) integrate over all beliefs b'. Given the complex nature of the belief space, it is not at all obvious that the integration can be carried out exactly, or that effective approximations can be found. It should therefore come at no surprise that calculating the value function V_T is more complicated in belief space than it is in state space.

Luckily, an exact solution exists for the interesting special case of finite worlds, in which the state space, the action space, the space of observations, and the planning horizon are all finite. This solution represents value functions by *piecewise linear functions* over the belief space. As we shall see, the linearity of this representation arises directly from the fact that the expectation is a linear operator. The piecewise nature is the result of the fact that the robot has the ability to select controls, and in different parts of the belief space it will select different controls. All these statements will be proven in this chapter.

PIECEWISE LINEAR FUNCTION

This chapter discusses the general POMDP algorithm for calculating policies defined over the space of all belief distributions. This algorithm is computationally cumbersome but correct for finite POMDPs; although a variant will be discussed that is highly tractable. The subsequent chapter will discuss a number of more efficient POMDP algorithms, which are approximate but scale to actual robotics problems.

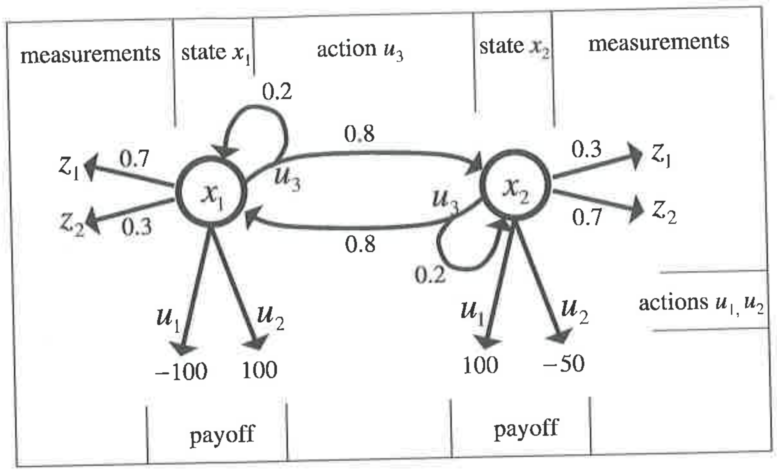


Figure 15.1 The two-state environment used to illustrate value iteration in belief space.

15.2 An Illustrative Example

15.2.1 Setup

We illustrate value iteration in belief spaces through a numerical example. This example is simplistic, but by discussing it we identify all major elements of value iteration in belief spaces.

Our example is the two-state world in Figure 15.1. The states are labeled x_1 and x_2. The robot can choose among three different control actions, u_1, u_2, and u_3. Actions u_1 and u_2 are terminal: When executed, they result in the following immediate payoff:

$$r(x_1, u_1) = -100 \qquad r(x_2, u_1) = +100 \tag{15.4}$$

$$r(x_1, u_2) = +100 \qquad r(x_2, u_2) = -50 \tag{15.5}$$

The dilemma is that both actions provide opposite payoffs in each of the states. Specifically, when in state x_1, u_2 is the optimal action, whereas it is u_1 in state x_2. Thus, knowledge of the state translates directly into payoff when selecting the optimal action.

To acquire such knowledge, the robot is provided with a third control action, u_3. Executing this control comes at a mild cost of -1:

$$r(x_1, u_3) = r(x_2, u_3) = -1 \tag{15.6}$$

One might think of this as the cost of waiting, or the cost of sensing. Action u_3 affects the state of the world in a non-deterministic manner:

(15.7) $$p(x'_1|x_1, u_3) = 0.2 \qquad p(x'_2|x_1, u_3) = 0.8$$

(15.8) $$p(x'_1|x_2, u_3) = 0.8 \qquad p(x'_2|x_2, u_3) = 0.2$$

In other words, when the robot executes u_3, the state flips to the respective other state with 0.8 probability, and the robot pays a unit cost.

Nevertheless, there is a benefit to executing action u_3. Before each control decision, the robot can sense. By sensing, the robot gains knowledge about the state, and in turn it can make a better control decision that leads to higher payoff in expectation. The action u_3 lets the robot sense without committing to a terminal action.

In our example, the measurement model is governed by the following probability distribution:

(15.9) $$p(z_1|x_1) = 0.7 \qquad p(z_2|x_1) = 0.3$$

(15.10) $$p(z_1|x_2) = 0.3 \qquad p(z_2|x_2) = 0.7$$

Put differently, if the robot measures z_1 its confidence increases for being in x_1, and the same is the case for z_2 relative to x_2.

The reason for selecting a two-state example is that it makes it easy to graph functions over the belief space. In particular, a belief state b is characterized by $p_1 = b(x_1)$ and $p_2 = b(x_2)$. However, we know $p_2 = 1 - p_1$, hence it suffices to graph p_1. The corresponding control policy π is a function that maps the unit interval $[0; 1]$ to the space of all actions:

(15.11) $$\pi : [0; 1] \longrightarrow u$$

15.2.2 Control Choice

In determining when to execute what control, let us start our consideration with the immediate payoff for each of the three control choices, u_1, u_2, and u_3. In the previous chapter, payoff was considered a function of state and actions. Since we do not know the state, we have to generalize the notion of a payoff to accommodate belief state. Specifically, for any given belief $b = (p_1, p_2)$, the expected payoff under this belief is given by the following expectation:

(15.12) $$r(b, u) = E_x[r(x, u)] = p_1\ r(x_1, u) + p_2\ r(x_2, u)$$

PAYOFF IN POMDPS The function $r(b, u)$ defines the *payoff in POMDPs*.

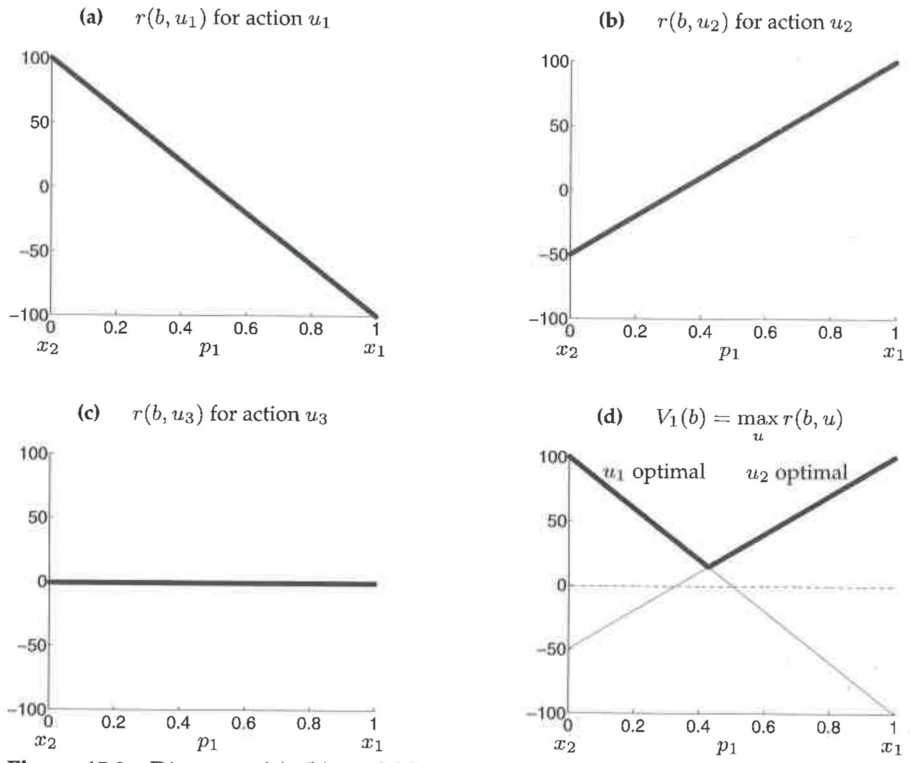


Figure 15.2 Diagrams (a), (b), and (c) depict the expected payoff r as a function of the belief state parameter $p_1 = b(x_1)$, for each of the three actions u_1, u_2, and u_3. (d) The value function at horizon $T = 1$ corresponds to the maximum of these three linear functions.

Figure 15.2a graphs the expected payoff $r(b, u_1)$ for control choice u_1, parameterized by the parameter p_1. On the leftmost end of this diagram, we have $p_1 = 0$, hence the robot believes the world to be in state x_2 with absolute confidence. Executing action u_1 hence yields $r(x_2, u_1) = 100$, as specified in Equation (15.4). On the rightmost end, we have $p_1 = 1$, hence the state is x_1. Consequently, control choice u_1 will result in $r(x_1, u_1) = -100$. In between, the expectation provides a linear combination of these two values:

$$r(b, u_1) \quad = \quad -100\, p_1 + 100\, p_2 \quad = \quad -100\, p_1 + 100\, (1 - p_1) \tag{15.13}$$

This function is graphed in Figure 15.2a.

Figures Figure 15.2b&c show the corresponding functions for action u_2 and

u_3, respectively. For u_2, we obtain

$$r(b, u_2) \quad = \quad 100\, p_1 - 50\, (1 - p_1) \tag{15.14}$$

and for u_3 we obtain the constant function

$$r(b, u_3) \quad = \quad -1\, p_1 - 1\, (1 - p_1) \quad = \quad -1 \tag{15.15}$$

Our first exercise in understanding value iteration in belief spaces will focus on the computation of the value function V_1, which is the value function that is optimal with regards to horizon $T = 1$ decision processes. Within a single decision cycle, our robot can choose among its three different control choices. So which one should it choose?

The answer is easily read off the diagrams studied thus far. For any belief state p_1, the diagrams in Figures 15.2a-c graph the expected payoff for each of the action choices. Since the goal is to maximize payoff, the robot simply selects the action of highest expected payoff. This is visualized in Figure 15.2d: This diagram superimposes all three expected payoff graphs. In the left region, u_1 is the optimal action, hence its value function dominates. The transition occurs when $r(b, u_1) = r(b, u_2)$, which resolves to $p_1 = \frac{3}{7}$. For values p_1 larger than $\frac{3}{7}$, u_2 will be the better action. Thus the $(T = 1)$-optimal policy is

$$\pi_1(b) \quad = \quad \begin{cases} u_1 & \text{if } p_1 \leq \frac{3}{7} \\ \\ u_2 & \text{if } p_1 > \frac{3}{7} \end{cases} \tag{15.16}$$

The corresponding value is then the thick upper graph in Figure 15.2d. This graph is a piecewise linear, convex function. It is the maximum of the individual payoff functions in Figures 15.2a-c. Thus, we can write it as a maximum over 3 functions:

$$\begin{aligned} V_1(b) \quad &= \quad \max_u \; r(b, u) \\ &= \quad \max \left\{ \begin{array}{ll} -100\, p_1 & +100\, (1 - p_1) \\ 100\, p_1 & -50\, (1 - p_1) \\ -1 & \end{array} \right\} \quad \begin{array}{l} (*) \\ (*) \\ \end{array} \end{aligned} \tag{15.17}$$

Obviously, only the linear functions marked $(*)$ in (15.17) contribute. The remaining linear function can safely be pruned away:

$$V_1(b) \quad = \quad \max \left\{ \begin{array}{ll} -100\, p_1 & +100\, (1 - p_1) \\ 100\, p_1 & -50\, (1 - p_1) \end{array} \right\} \tag{15.18}$$

We will use this pruning trick repeatedly in our example. Prunable linear constraints are shown as dashed lines in Figure 15.2d and many graphs to follow.

15.2.3 Sensing

The next step in our reasoning involves perception. What if the robot can sense before it chooses its control? How does this affect the optimal value function? Obviously, sensing provides information about the state, hence should enable the robot to choose a better control action. Specifically, for the worst possible belief thus far, $p_1 = \frac{3}{7}$, the expected payoff in our example was $\frac{100}{7} \approx 14.3$, which is the value at the kink in Figure 15.2d. Clearly, if we can sense first, we find ourselves in a different belief after sensing. The value of this belief will be better than 14.3, but by how much?

The answer is surprising. Suppose we sense z_1. Figure 15.3a shows the belief after sensing z_1 as a function of the belief before sensing. Let us dissect this function. If our pre-sensing belief is $p_1 = 0$, our post-sensing belief is also $p_1 = 0$, regardless of the measurement. Similarly for $p_1 = 1$. Hence, at the extreme ends, this function is the identity. In between, we are uncertain as to what the state of the world is, and measuring z_1 does shift our belief. The amount it shifts is governed by Bayes rule:

(15.19)
$$\begin{aligned} p'_1 &= p(x_1 \mid z) \\ &= \frac{p(z_1 \mid x_1)\, p(x_1)}{p(z_1)} \\ &= \frac{0.7\, p_1}{p(z_1)} \end{aligned}$$

and

(15.20)
$$p'_2 = \frac{0.3\,(1 - p_1)}{p(z_1)}$$

The normalizer $p(z_1)$ adds the non-linearity in Figure 15.3a. In our example, it resolves to

(15.21)
$$p(z_1) = 0.7\, p_1 + 0.3\,(1 - p_1) = 0.4\, p_1 + 0.3$$

and hence $p'_1 = \frac{0.7\, p_1}{0.4\, p_1 + 0.3}$. However, as we shall see below, this normalizer nicely cancels out. More on this in a minute.

Let us first study the effect of this non-linear transfer function on the value function V_1. Suppose we know that we observed z_1, and then have

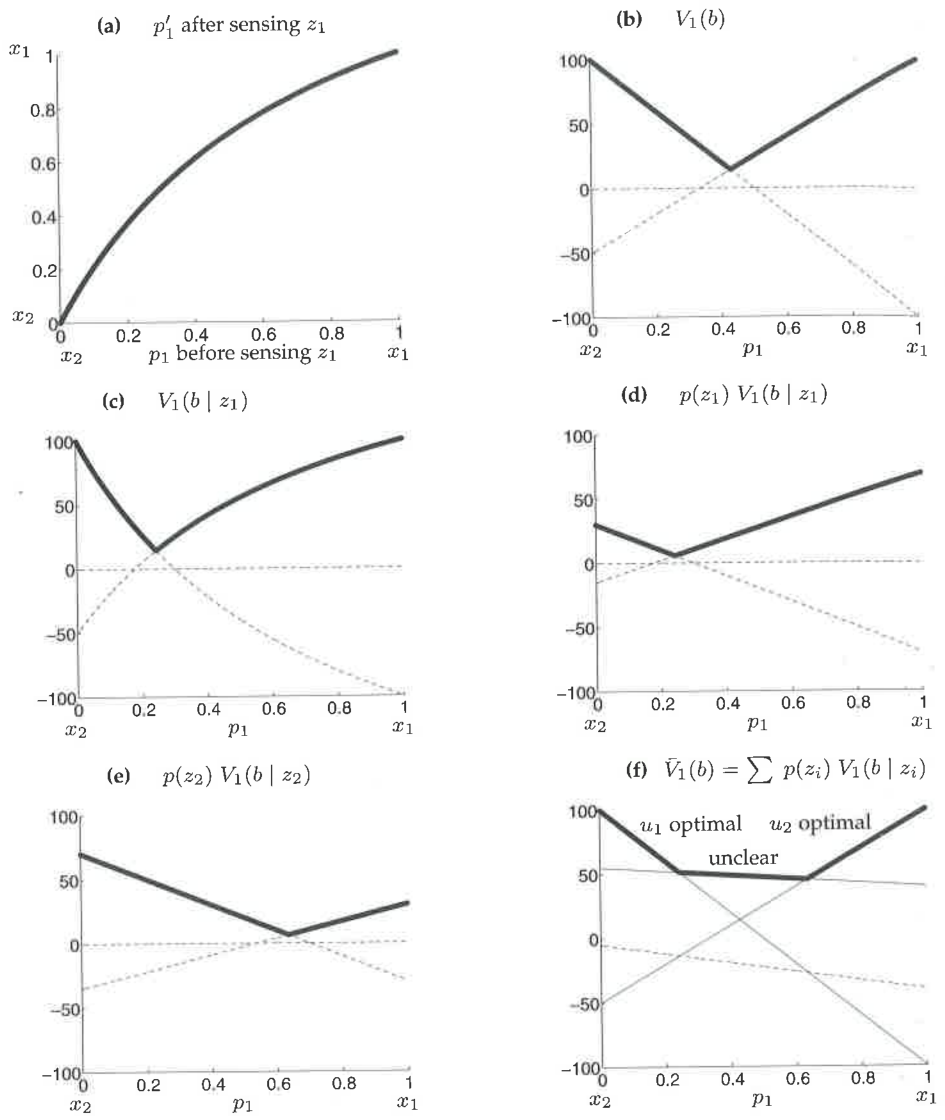


Figure 15.3 The effect of sensing on the value function: (a) The belief after sensing z_1 as a function of the belief before sensing z_1. Sensing z_1 makes the robot more confident that the state is x_1. Projecting the value function in (b) through this nonlinear function results in the non-linear value function in (c). (d) Dividing this value function by the probability of observing z_1 results in a piecewise linear function. (e) The same piecewise linear function for measurement z_2. (f) The expected value function after sensing.

to make an action choice. What would that choice be, and what would the corresponding value function look like? The answer is given graphically in Figure 15.3c. This figure depicts the piecewise linear value function in Figure 15.3b, mapped through the nonlinear measurement function discussed above (and shown in Figure 15.3a). The reader may take a moment to get oriented here: Take a belief p_1, map it to a corresponding belief p'_1 according to our non-linear function, and then read off its value in Figure 15.3b. This procedure, for all $p_1 \in [0; 1]$, leads to the graph in Figure 15.3c.

Mathematically, this graph is given by

$$V_1(b \mid z_1) \;=\; \max \left\{ \begin{array}{ll} -100 \cdot \frac{0.7\,p_1}{p(z_1)} & +100 \cdot \frac{0.3\,(1-p_1)}{p(z_1)} \\ 100 \cdot \frac{0.7\,p_1}{p(z_1)} & -50 \cdot \frac{0.3\,(1-p_1)}{p(z_1)} \end{array} \right\} \tag{15.22}$$

$$= \frac{1}{p(z_1)} \max \left\{ \begin{array}{ll} -70\,p_1 & +30\,(1-p_1) \\ 70\,p_1 & -15\,(1-p_1) \end{array} \right\}$$

which is simply the result of replacing p_1 by p'_1 in the value function V_1 specified in (15.18). We note in Figure 15.3c that the belief of "worst" value has shifted to the left. Now the worst belief is the one that, after sensing z_1, makes us believe with $\frac{3}{7}$ probability we are in state x_1.

However, this is the consideration for one of the two measurements only, the value before sensing has to take both measurements into account. Specifically, the value before sensing, denoted $\bar{V}_1$, is given by the following expectation:

$$\bar{V}_1(b) \;=\; E_z[V_1(b \mid z)] \;=\; \sum_{i=1}^{2} p(z_i)\, V_1(b \mid z_i) \tag{15.23}$$

We immediately notice that in this expectation, each contributing value function $V_1(b \mid z_i)$ is multiplied by the probability $p(z_i)$, which was the cause of the nonlinearity in the pre-measurement value function. Plugging (15.19) into this expression gives us

$$\bar{V}_1(b) \;=\; \sum_{i=1}^{2} p(z_i)\, V_1\!\left(\frac{p(z_i \mid x_1)\, p_1}{p(z_i)}\right) \tag{15.24}$$

$$= \sum_{i=1}^{2} p(z_i)\, \frac{1}{p(z_i)}\, V_1(p(z_i \mid x_1)\, p_1)$$

$$= \sum_{i=1}^{2} V_1(p(z_i \mid x_1)\, p_1)$$

This transformation is true because each element in V_1 is linear in $1/p(z_i)$, as illustrated by example in (15.22). There we were able to move the factor $1/p(z_i)$ out of the maximization, since each term in the maximization is a product of this factor. After restoring the terms accordingly, the term $p(z_i)$ simply cancels out!

In our example, we have two measurements, hence we can compute the expectation $p(z_i)\ V_1(b \mid z_i)$ for each of these measurements. The reader may recall that these terms are added in the expectation (15.23). For z_1, we already computed $V_1(b \mid z_i)$ in (15.22), hence

(15.25) $$p(z_1)\ V_1(b \mid z_1) \quad = \quad \max \left\{ \begin{array}{rr} -70\, p_1 & +30\,(1-p_1) \\ 70\, p_1 & -15\,(1-p_1) \end{array} \right\}$$

This function is shown in Figure 15.3d: It is indeed the maximum of two linear functions. Similarly, for z_2 we obtain

(15.26) $$p(z_2)\ V_1(b \mid z_2) \quad = \quad \max \left\{ \begin{array}{rr} -30\, p_1 & +70\,(1-p_1) \\ 30\, p_1 & -35\,(1-p_1) \end{array} \right\}$$

This function is depicted in Figure 15.3e.

The desired value function before sensing is then obtained by adding those two terms, according to Equation (15.23):

(15.27) $$\bar{V}_1 \quad = \quad \max \left\{ \begin{array}{rr} -70\, p_1 & +30\,(1-p_1) \\ 70\, p_1 & -15\,(1-p_1) \end{array} \right\} + \max \left\{ \begin{array}{rr} -30\, p_1 & +70\,(1-p_1) \\ 30\, p_1 & -35\,(1-p_1) \end{array} \right\}$$

This sum is shown in Figure 15.3f. It has a remarkable shape: Instead of a single kink, it possesses two different kinks, separating the value function into three different linear segments. For the left segment, u_1 is the optimal action, no matter what additional information the robot may reap through future sensing. Similarly for the right segment, u_2 is the optimal control action no matter what. In the center region, however, sensing matters. The optimal action is determined by what the robot senses. In doing so, the center segment defines a value that is significantly higher than the corresponding value without sensing, shown in Figure 15.2d. Essentially, the ability to sense lifted an entire region in the value function to a higher level, in the region where the robot was least certain about the state of the world. This remarkable finding shows that value iteration in belief space indeed values sensing, but only to the extent that it matters for future control choices.

Let us return to computing this value function, since it may appear easier than it is. Equation (15.27) requires us to compute the sum of two maxima over linear functions. Bringing this into our canonical form—which is the maximum over linear functions without the sum—requires some thought. Specifically, our new value function $\bar{V}_1$ will be bounded below by *any* sum that adds a linear function from the first max-expression to a linear function from the second max-expression. This leaves us with four possible combinations:

(15.28)
$$\begin{aligned}
\bar{V}_1(b) &= \max \left\{ \begin{array}{llll} -70\,p_1 & +30\,(1-p_1) & -30\,p_1 & +70\,(1-p_1) \\ -70\,p_1 & +30\,(1-p_1) & +30\,p_1 & -35\,(1-p_1) \\ 70\,p_1 & -15\,(1-p_1) & -30\,p_1 & +70\,(1-p_1) \\ 70\,p_1 & -15\,(1-p_1) & +30\,p_1 & -35\,(1-p_1) \end{array} \right\} \\
&= \max \left\{ \begin{array}{ll} -100\,p_1 & +100\,(1-p_1) \\ -40\,p_1 & -5\,(1-p_1) \\ 40\,p_1 & +55\,(1-p_1) \\ 100\,p_1 & -50\,(1-p_1) \end{array} \right\} \begin{array}{c} (*) \\ \\ (*) \\ (*) \end{array} \\
&= \max \left\{ \begin{array}{ll} -100\,p_1 & +100\,(1-p_1) \\ 40\,p_1 & +55\,(1-p_1) \\ 100\,p_1 & -50\,(1-p_1) \end{array} \right\}
\end{aligned}$$

Once again, we use $(*)$ to denote constraints that actually contribute to the definition of the value function. As shown in Figure 15.3f, only three of these four linear functions are required, and the fourth can safely be pruned away.

15.2.4 Prediction

Our final step concerns state transitions. When the robot selects an action, its state changes. To plan at a horizon larger than $T = 1$, we have to take this into consideration and project our value function accordingly. In our example, u_1 and u_2 are both terminal actions. Thus, we only have to consider the effect of action u_3.

Luckily, state transitions are not anywhere as intricate as measurements in POMDPs. Figure 15.4a shows mapping of the belief upon executing u_3. Specifically, suppose we start out in state x_1 with absolute certainty, hence $p_1 = 1$. Then according to our transition probability model in Equation (15.7), we have $p'_1 = p(x'_1|x_1, u_3) = 0.2$. Similarly, for $p_1 = 0$ we get $p'_1 = p(x'_1|x_2, u_3) = 0.8$. In between the expectation is linear:

(15.29)
$$\begin{aligned}
p'_1 &= E_x[p(x'_1|x, u_3)] \\
&= \sum_{i=1}^{2} p(x'_1|x_i, u_3)\, p_i
\end{aligned}$$

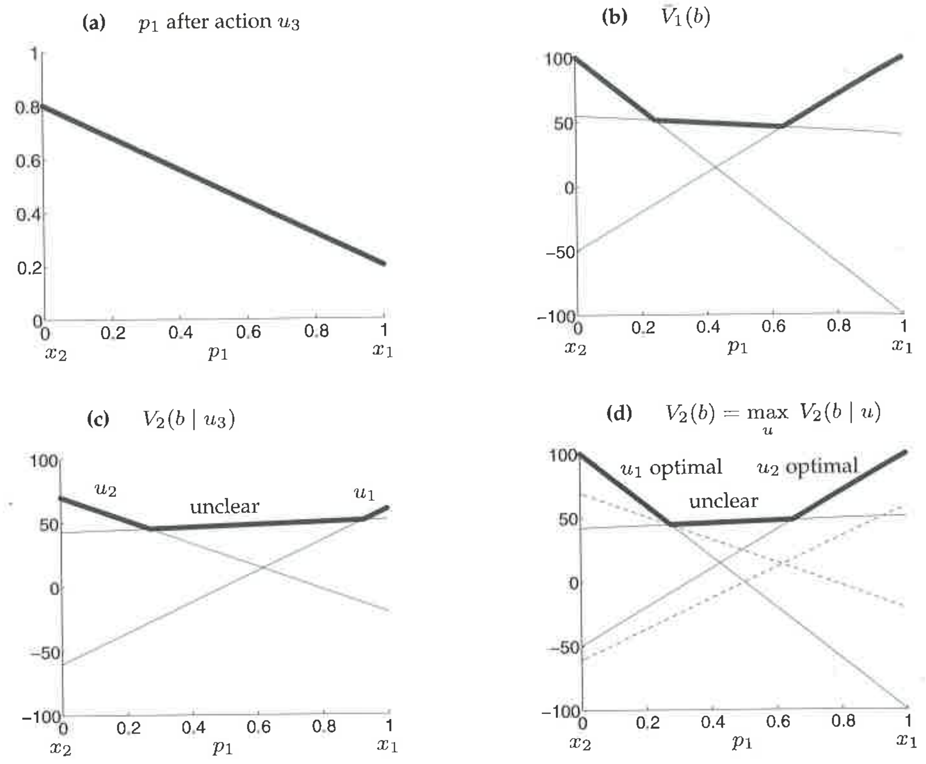


Figure 15.4 (a) The belief state parameter p_1' after executing action u_3, as a function of the parameter p_1 before the action. Propagating the belief shown in (b) through the inverse of this mapping results in the belief shown in (c). (d) The value function V_2 obtained by maximizing the propagated belief function, and the payoff of the two remaining actions, u_1 and u_2.

$$= \quad 0.2\, p_1 + 0.8\, (1 - p_1) \quad = \quad 0.8 - 0.6\, p_1$$

This is the function graphed in Figure 15.4a. If we now back-project the value function in Figure 15.4b—which is equivalent to the one shown in Figure 15.3f—we obtain the value function in Figure 15.4c. This value function is flatter than the one before the projection step, reflecting the loss of information through the state transition. It is also mirrored, since in expectation the state changes when executing u_3.

Mathematically, this value function is computed by projecting (15.28) through (15.29).

$$\bar{V}_1(b \mid u_3) \;=\; \max \left\{ \begin{array}{ll} -100\,(0.8-0.6\,p_1) & +100\,(1-(0.8-0.6\,p_1)) \\ 40\,(0.8-0.6\,p_1) & +55\,(1-(0.8-0.6\,p_1)) \\ 100\,(0.8-0.6\,p_1) & -50\,(1-(0.8-0.6\,p_1)) \end{array} \right\} \tag{15.30}$$

$$= \max \left\{ \begin{array}{ll} -100\,(0.8-0.6\,p_1) & +100\,(0.2+0.6\,p_1) \\ 40\,(0.8-0.6\,p_1) & +55\,(0.2+0.6\,p_1) \\ 100\,(0.8-0.6\,p_1) & -50\,(0.2+0.6\,p_1) \end{array} \right\}$$

$$= \max \left\{ \begin{array}{ll} 60\,p_1 & -60\,(1-p_1) \\ 52\,p_1 & +43\,(1-p_1) \\ -20\,p_1 & +70\,(1-p_1) \end{array} \right\}$$

These transformations are easily checked by hand. Figure 15.4c shows this function, along with the optimal control actions.

We have now almost completed the vale function V_2 with a planning horizon of $T = 2$. Once again, the robot is given a choice whether to execute the control u_3, or to directly engage in any of the terminal actions u_1 or u_2. As before, this choice is implemented by adding two new options to our consideration, in the form of the two linear functions $r(b, u_1)$ and $r(b, u_2)$. We also must subtract the cost of executing action u_3 from the value function.

This leads to the diagram in Figure 15.4d, which is of the form

$$\bar{V}_2(b) \;=\; \max \left\{ \begin{array}{ll} -100\,p_1 & +100\,(1-p_1) \\ 100\,p_1 & -50\,(1-p_1) \\ 59\,p_1 & -61\,(1-p_1) \\ 51\,p_1 & +42\,(1-p_1) \\ -21\,p_1 & +69\,(1-p_1) \end{array} \right\} \begin{array}{l} (*) \\ (*) \\ \\ (*) \\ \\ \end{array} \tag{15.31}$$

Notice that we simply added the two options (lines one and two), and subtracted the uniform cost of u_3 from all other linear constraints (lines three through five). Once again, only three of those constraints are needed, as indicated by the $(*)$'s. The resulting value can thus be rewritten as

$$\bar{V}_2(b) \;=\; \max \left\{ \begin{array}{ll} -100\,p_1 & +100\,(1-p_1) \\ 100\,p_1 & -50\,(1-p_1) \\ 51\,p_1 & +42\,(1-p_1) \end{array} \right\} \tag{15.32}$$

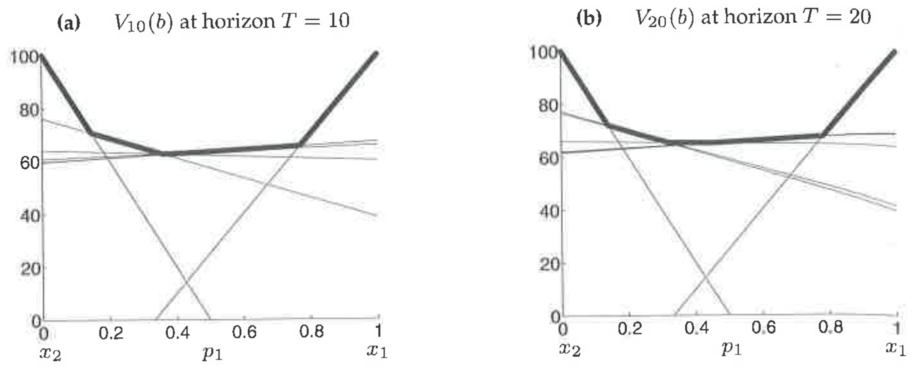


Figure 15.5 The value function V for horizons $T = 10$ and $T = 20$. Note that the vertical axis in these plots differs in scale from previous depictions of value functions.

15.2.5 Deep Horizons and Pruning

BACKUP STEP IN BELIEF SPACE

We have now executed a full *backup step in belief space*. This algorithm is easily recursed. Figure 15.5 shows the value function at horizon $T = 10$ and $T = 20$, respectively. Both of these value functions are seemingly similar. With appropriate pruning, V_{20} has only 13 components

$$\bar{V}_{20}(b) \quad = \quad \max \left\{ \begin{array}{rr} -100\, p_1 & +100\,(1-p_1) \\ 100\, p_1 & -50\,(1-p_1) \\ 64.1512\, p_1 & +65.9454\,(1-p_1) \\ 64.1513\, p_1 & +65.9454\,(1-p_1) \\ 64.1531\, p_1 & +65.9442\,(1-p_1) \\ 68.7968\, p_1 & +62.0658\,(1-p_1) \\ 68.7968\, p_1 & +62.0658\,(1-p_1) \\ 69.0914\, p_1 & +61.5714\,(1-p_1) \\ 68.8167\, p_1 & +62.0439\,(1-p_1) \\ 69.0369\, p_1 & +61.6779\,(1-p_1) \\ 41.7249\, p_1 & +76.5944\,(1-p_1) \\ 39.8427\, p_1 & +77.1759\,(1-p_1) \\ 39.8334\, p_1 & +77.1786\,(1-p_1) \end{array} \right\} \tag{15.33}$$

We recognize the two familiar linear functions on the top; all others correspond to specific sequences of measurements and action choices.

As simple consideration shows that pruning is of essence. Without pruning, each update brings two new linear constraints (action choice), and then squares the number of constraints (measurement). Thus, an unpruned value function for $T = 20$ is defined over $10^{547,864}$ linear functions; at $T = 30$ we have $10^{561,012,337}$ linear constraints. The pruned value function, in comparison, contains only 13 such constraints.

This enormous explosion of linear pieces is a key reason why plain POMDPs are impractical. Figure 15.6 compares side-by-side the steps that led to the value function V_2. The left column shows our pruned functions, whereas the right row maintains all linear functions without pruning. While we only have a single measurement update in this calculation, the number of unused functions is already enormous. We will return to this point later, when we will devise efficient approximate POMDP algorithms.

A final observation of our analysis is that the optimal value function for any finite horizon is continuous, piecewise linear, and convex. Each linear piece corresponds to a different action choice at some point in the future. The convexity of the value function indicates the rather intuitive observation, namely that knowing is always superior to not knowing. Given two belief states b and b', the mixed value of the belief states is larger or equal to the value of the mixed belief state, for some mixing parameter β with $0 \leq \beta \leq 1$:

$$\beta V(b) + (1-\beta)V(b') \quad \geq \quad V(\beta b + (1-\beta)b') \tag{15.34}$$

This characterization only applies to the finite horizon case. Under infinite horizon, the value function can be discontinuous and nonlinear.

15.3 The Finite World POMDP Algorithm

The previous section showed, by example, how to calculate value functions in finite worlds. Here we briefly discuss a general algorithm for calculating a value function, before deriving it from first principles.

The algorithm **POMDP** is listed in Table 15.1. This algorithm accepts as an input just a single parameter: T, the planning horizon for the POMDP. It returns a set of parameter vectors, each of the form

$$(v_1, \ldots, v_N) \tag{15.35}$$

Each of these parameters specifies a linear function over the belief space of the form

$$\sum_i v_i \, p_i \tag{15.36}$$

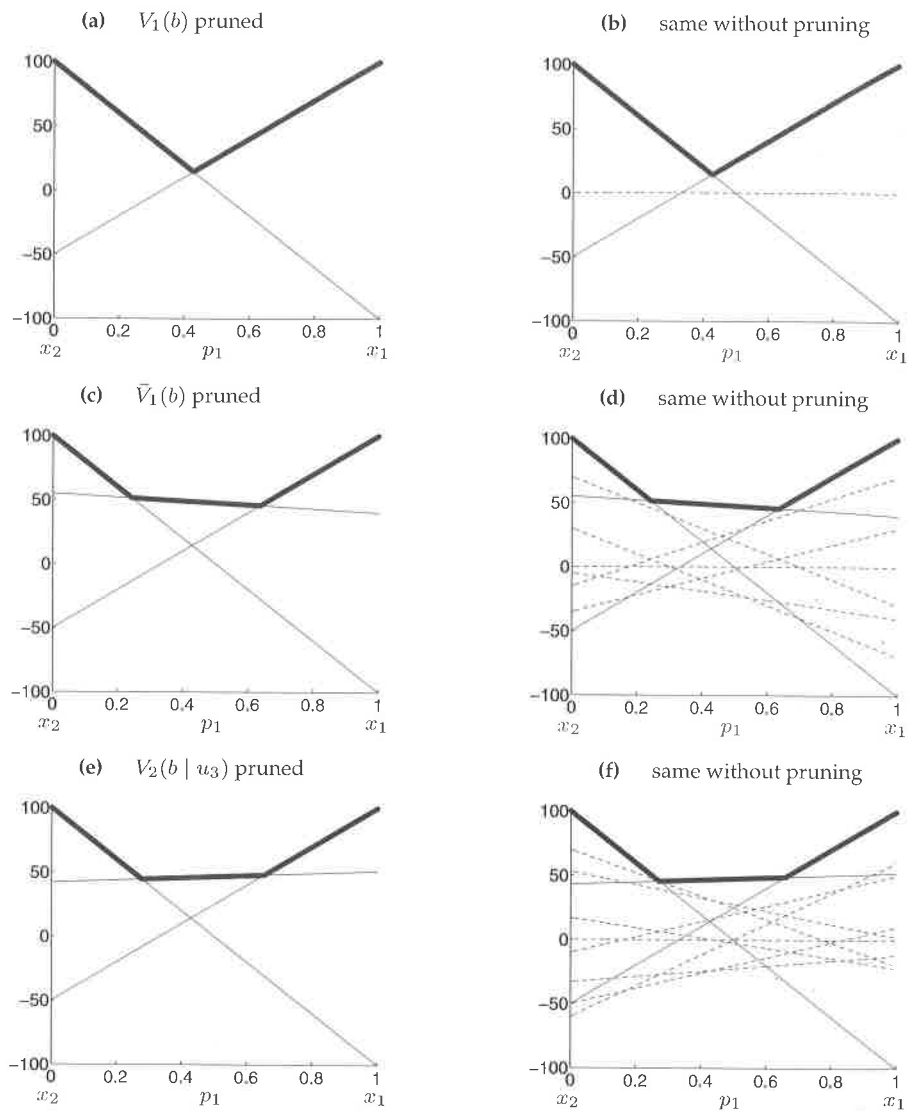


Figure 15.6 Comparison of an exact pruning algorithm (left row) versus a non-pruning POMDP algorithm (right row), for the first few steps of the POMDP planning algorithm. Obviously, the number of linear constraints increases dramatically without pruning. At $T = 20$, the unpruned value function is defined over $10^{547,864}$ linear functions, whereas the pruned one only uses 13 such functions.

```
Algorithm POMDP($T$):
    $\Upsilon = (0; 0, \ldots, 0)$
    for $\tau = 1$ to $T$ do
        $\Upsilon' = \emptyset$
        for all $(u'; v_1^k, \ldots, v_N^k)$ in $\Upsilon$ do
            for all control actions $u$ do
                for all measurements $z$ do
                    for $j = 1$ to $N$ do
                        $v_{u,z,j}^k = \sum_{i=1}^{N} v_i^k \; p(z \mid x_i) \; p(x_i \mid u, x_j)$
                    endfor
                endfor
            endfor
        endfor
        for all control actions $u$ do
            for all $k(1), \ldots, k(M) = (1, \ldots, 1)$ to $(|\Upsilon|, \ldots, |\Upsilon|)$ do
                for $i = 1$ to $N$ do
                    $v_i' = \gamma \left[ r(x_i, u) + \sum_z v_{u,z,i}^{k(z)} \right]$
                endfor
                add $(u; v_1', \ldots, v_N')$ to $\Upsilon'$
            endfor
        endfor
        optional: prune $\Upsilon'$
        $\Upsilon = \Upsilon'$
    endfor
    return $\Upsilon$
```

Table 15.1 The POMDP algorithm for discrete worlds. This algorithm represents the optimal value function by a set of linear constraints, which are calculated recursively.

```
1:      Algorithm policy_POMDP($\Upsilon, b = (p_1, \ldots, p_N)$):

2:          $\hat{u} = \underset{(u; v_1^k, \ldots, v_N^k) \in \Upsilon}{\text{argmax}} \sum_{i=1}^{N} v_i^k \, p_i$

3:          return $\hat{u}$
```

Table 15.2 The algorithm for determining the optimal action for a policy represented by the set of linear functions Υ.

MAXIMUM OF LINEAR FUNCTIONS

The actual value is governed by the *maximum of all these linear functions*:

$$\max_{(p_1,\ldots,p_N)} \sum_i v_i \, p_i \tag{15.37}$$

The algorithm **POMDP** computes this value function recursively. An initial set for the pseudo-horizon $T = 0$ is set in line 2 of Table 15.1. The algorithm **POMDP** then recursively computes a new set in the nested loop of lines 3-24. A key computational step takes place in line 9: Here, the coefficients $v_{u,z,j}^k$ of the linear functions needed to compute the next set of linear constraints are computed. Each linear function results from executing control u, followed by observing measurement z, and then executing control u'. The linear constraint corresponding to u' was calculated in the previous iteration for a smaller planning horizon (taken in line 5). Thus, upon reaching line 14, the algorithm has generated one linear function for each combination of control action, measurement, and linear constraint of the previous value function.

The linear constraints of the new value function result by taking the expectations over measurements, as done in lines 14-21. For each control action, the algorithm generates K^M such linear constraints in line 15. This large number is due to the fact that each expectation is taken over the M possible measurements, each of which can be "combined" with any of the K constraints contained in the previous value function. Line 17 computes the expectation for each such combination. The resulting constraint is added to the new set of constraints in line 19.

The algorithm for finding the optimal control action is shown in Table 15.2. The input to this algorithm is a belief state, parameterized by

$b = (p_1, \ldots, p_N)$, along with the set of linear functions Υ. The optimal action is determined by search through all linear functions, and identifying the one that maximizes its value for b. This value is returned in line 3 of the algorithm **policy_POMDP:**, Table 15.2.

15.4 Mathematical Derivation of POMDPs

15.4.1 Value Iteration in Belief Space

The general update for the value function implements (15.2), restated here for convenience.

$$V_T(b) \quad = \quad \gamma \max_u \left[r(b,u) + \int V_{T-1}(b')\, p(b' \mid u, b)\, db' \right] \tag{15.38}$$

We begin by transforming this equation into a more practical form, one that avoids integration over the space of all possible beliefs.

A key factor in this update is the conditional probability $p(b' \mid u, b)$. This probability specifies a distribution over probability distributions. Given a belief b and a control action u, the outcome is indeed a distribution over distributions. This is because the concrete belief b' is also based on the next measurement, the measurement itself is generated stochastically. Dealing with distributions of distributions adds an element of complexity that is undesirable.

If we fix the measurement, the posterior b' is unique and $p(b' \mid u, b)$ degenerate to a point-mass distribution. Why is this so? The answer is provided by the Bayes filter. From the belief b before action execution, the action u, and the subsequent observation z, the Bayes filter calculates a single, posterior belief b' which is the single, correct belief. Thus, we conclude that if only we knew z, the integration over all beliefs in (15.38) would be obsolete.

This insight can be exploited by re-expressing

$$p(b' \mid u, b) \quad = \quad \int p(b' \mid u, b, z)\, p(z \mid u, b)\, dz \tag{15.39}$$

where $p(b' \mid u, b, z)$ is a point-mass distribution focused on the single belief calculated by the Bayes filter. Plugging this integral into Equation (15.38) gives us

$$V_T(b) \quad = \quad \gamma \max_u \left[r(b,u) + \int \left[\int V_{T-1}(b')\, p(b' \mid u, b, z)\, db' \right] p(z \mid u, b)\, dz \right] \tag{15.40}$$

The inner integral

(15.41)
$$\int V_{T-1}(b')\; p(b' \mid u, b, z)\; db'$$

contains only one non-zero term. This is the term where b' is the distribution calculated from b, u, and z using the Bayes filter. Let us call this distribution $B(b, u, z)$:

(15.42)
$$\begin{aligned} B(b,u,z)(x') &= p(x' \mid z, u, b) \\ &= \frac{p(z \mid x', u, b)\; p(x' \mid u, b)}{p(z \mid u, b)} \\ &= \frac{1}{p(z \mid u, b)}\; p(z \mid x') \int p(x' \mid u, b, s)\; p(x \mid u, b)\; dx \\ &= \frac{1}{p(z \mid u, b)}\; p(z \mid x') \int p(x' \mid u, x)\; b(x)\; dx \end{aligned}$$

The reader should recognize the familiar Bayes filter derivation that was extensively discussed in Chapter 2, this time with the normalizer made explicit.

We can now rewrite (15.40) as follows. Note that this expression no longer integrates over b'.

(15.43)
$$V_T(b) \;=\; \gamma \max_u \left[r(b,u) + \int V_{T-1}(B(b,u,z))\; p(z \mid u, b)\; dz \right]$$

This form is more convenient than the original one in (15.38), since it only requires integration over all possible measurements z, instead of all possible belief distributions b'. This transformation was used implicitly in the example above, where a new value function was obtained by mixing together finitely many piecewise linear functions.

Below, it will be convenient to split the maximization over actions from the integration. Hence, we notice that (15.43) can be rewritten as the following two equations:

(15.44)
$$V_T(b,u) \;=\; \gamma \left[r(b,u) + \int V_{T-1}(B(b,u,z))\; p(z \mid u, b)\; dz \right]$$

(15.45)
$$V_T(b) \;=\; \max_u V_T(b,u)$$

Here $V_T(b,u)$ is the horizon T-value function over the belief b, assuming that the immediate next action is u.

15.4.2 Value Function Representation

As in our example, we represent the value function by a maximum of a set of linear functions. We already discussed that any linear function over the belief simplex can be represented by the set of coefficients $v_1, \ldots, v_N$:

$$V(b) \quad = \quad \sum_{i=1}^{N} v_i \, p_i \tag{15.46}$$

where, as usual, $p_1, \ldots, p_N$ are the parameters of the belief distribution b. As in our example, a piecewise linear and convex value function $V_T(b)$ can be represented by the maximum of a finite set of linear functions

$$V(b) \quad = \quad \max_k \sum_{i=1}^{N} v_i^k \, p_i \tag{15.47}$$

where $v_1^k, \ldots, v_N^k$ denotes the parameters of the k-th linear function. The reader should quickly convince herself that the maximum of a finite set of linear functions is indeed a convex, continuous, and piecewise linear function.

15.4.3 Calculating the Value Function

We will now derive a recursive equation for calculating the value function $V_T(b)$. We assume by induction that $V_{T-1}(b)$, the value function for horizon $T-1$, is represented by a piecewise linear function as specified above. As part of the derivation, we will show that under the assumption that $V_{T-1}(b)$ is piecewise linear and convex, $V_T(b)$ is also piecewise linear and convex. Induction over the planning horizon T then proves that all value functions with finite horizon are indeed piecewise linear and convex.

We begin with Equations (15.44) and (15.45). If the measurement space is finite, we can replace the integration over z by a finite sum.

$$V_T(b,u) \quad = \quad \gamma \left[r(b,u) + \sum_z V_{T-1}(B(b,u,z)) \, p(z \mid u,b) \right] \tag{15.48}$$

$$V_T(b) \quad = \quad \max_u V_T(b,u) \tag{15.49}$$

The belief $B(b,u,z)$ is obtained using the following expression, derived from Equation (15.42) by replacing the integral with a finite sum.

$$B(b,u,z)(x') \quad = \quad \frac{1}{p(z \mid u,b)} \, p(z \mid x') \sum_x p(x' \mid u,x) \, b(x) \tag{15.50}$$

If the belief b is represented by the parameters $\{p_1, \ldots, p_N\}$, and the belief $B(b,u,z)$ by $\{p'_1, \ldots, p'_N\}$, it follows that the j-th parameter of the belief b' is computed as follows:

(15.51) $$p'_j = \frac{1}{p(z \mid u,b)}\, p(z \mid x_j) \sum_{i=1}^{N} p(x_j \mid u, x_i)\, p_i$$

To compute the value function update (15.48), let us now find more practical expressions for the term $V_{T-1}(B(b,u,z))$, using the finite sums described above. Our derivation starts with the definition of V_{T-1} and substitutes the p'_j according to Equation (15.51):

(15.52) $$\begin{aligned} V_{T-1}(B(b,u,z)) &= \max_k \sum_{j=1}^{N} v_j^k\, p'_j \\ &= \max_k \sum_{j=1}^{N} v_j^k\, \frac{1}{p(z \mid u,b)}\, p(z \mid x_j) \sum_{i=1}^{N} p(x_j \mid u, x_i)\, p_i \\ &= \frac{1}{p(z \mid u,b)} \max_k \sum_{j=1}^{N} v_j^k\, p(z \mid x_j) \sum_{i=1}^{N} p(x_j \mid u, x_i)\, p_i \\ &= \frac{1}{p(z \mid u,b)} \max_k \overbrace{\sum_{i=1}^{N} p_i \underbrace{\sum_{j=1}^{N} v_j^k\, p(z \mid x_j)\, p(x_j \mid u, x_i)}_{(*)}}^{(**)} \end{aligned}$$

The term marked $(*)$ is independent of the belief. Hence, the function labeled $(**)$ is a linear function in the parameters of the belief space, $p_1, \ldots, p_N$. The term $1/p(z \mid u,b)$ is both nonlinear and difficult to compute, since it contains an entire belief b as conditioning variable. However, the beauty of POMDPs is that this expression cancels out. In particular, substituting this expression back into (15.48) yields the following update equation:

(15.53) $$V_T(b,u) = \gamma \left[r(b,u) + \sum_z \max_k \sum_{i=1}^{N} p_i \sum_{j=1}^{N} v_j^k\, p(z \mid x_j)\, p(x_j \mid u, x_i) \right]$$

Hence, despite the non-linearity arising from the measurement update, $V_T(b,u)$ is once again piecewise linear.

Finally, we note that $r(b,u)$ is given by the expectation

(15.54) $$r(b,u) = E_x[r(x,u)] = \sum_{i=1}^{N} p_i\, r(x_i, u)$$

Here we assumed that the belief b is represented by the parameters $\{p_1, \ldots, p_N\}$.

The desired value function V_T is now obtained by maximizing $V_T(b, u)$ over all actions u, as stated in (15.49):

(15.55)
$$\begin{aligned}
V_T(b) &= \max_u V_T(b, u) \\
&= \gamma \max_u \left(\left[\sum_{i=1}^N p_i\, r(x_i, u) \right] + \sum_z \max_k \right. \\
&\qquad\qquad \left. \sum_{i=1}^N p_i \underbrace{\sum_{j=1}^N v_j^k\, p(z \mid x_j)\, p(x_j \mid u, x_i)}_{=:\, v_{u,z,i}^k} \right) \\
&= \gamma \max_u \left(\left[\sum_{i=1}^N p_i\, r(x_i, u) \right] + \underbrace{\sum_z \max_k \sum_{i=1}^N p_i\, v_{u,z,i}^k}_{(*)} \right)
\end{aligned}$$

with

(15.56)
$$v_{u,z,i}^k = \sum_{j=1}^N v_j^k\, p(z \mid x_j)\, p(x_j \mid u, x_i)$$

as indicated. This expression is not yet in the form of a maximum of linear functions. In particular, we now need to change the sum-max-sum expression labeled $(*)$ in (15.55) into a max-sum-sum expression, which is the familiar form of a maximum over a set of linear functions.

We utilize the same transformation as in our example, Chapter 15.2.3. Specifically, suppose we would like to compute the maximum

(15.57)
$$\max\{a_1(x), \ldots, a_n(x)\} + \max\{b_1(x), \ldots, b_n(x)\}$$

for some functions $a_1(x), \ldots, a_n(x)$ and $b_1(x), \ldots, b_n(x)$ over a variable x. This maximum is attained at

(15.58)
$$\max_i \max_j\ [a_i(x) + b_j(x)]$$

This follows from the fact that each $a_i + b_j$ is indeed a lower bound. Further for any x there must exist an i and j such that $a_i(x)+b_j(x)$ defines the maximum. By including all such potential pairs in (15.58) we obtain a tight lower bound, i.e., the solution.

This is now easily generalized into arbitrary sums over max expressions:

(15.59) $$\sum_{j=1}^{m} \max_{i=1}^{N} a_{i,j}(x) \quad = \quad \max_{i(1)=1}^{N} \max_{i(2)=1}^{N} \cdots \max_{i(m)=1}^{N} \sum_{j=1}^{m} a_{i(j),j}$$

We apply now this "trick" to our POMDP value function calculation and obtain for the expression (∗) in (15.55). Let M be the total number of measurements.

(15.60) $$\begin{aligned} \sum_{z} \max_{k} \sum_{i=1}^{N} p_i \, v_{u,z,i}^{k} &= \max_{k(1)} \max_{k(2)} \cdots \max_{k(M)} \sum_{z} \sum_{i=1}^{N} p_i \, v_{u,z,i}^{k(z)} \\ &= \max_{k(1)} \max_{k(2)} \cdots \max_{k(M)} \sum_{i=1}^{N} p_i \sum_{z} v_{u,z,i}^{k(z)} \end{aligned}$$

Here each $k(\,)$ is a separate variable, each of which takes on the values of the variable k on the left hand side. There are as many such variables as there are measurements. As a result, the desired value function is now obtained as follows:

(15.61) $$\begin{aligned} V_T(b) &= \gamma \max_{u} \left[\sum_{i=1}^{N} p_i \, r(x_i, u) \right] + \max_{k(1)} \max_{k(2)} \cdots \max_{k(M)} \sum_{i=1}^{N} p_i \sum_{z} v_{u,z,i}^{k(z)} \\ &= \gamma \max_{u} \max_{k(1)} \max_{k(2)} \cdots \max_{k(M)} \sum_{i=1}^{N} p_i \left[r(x_i, u) + \sum_{z} v_{u,z,i}^{k(z)} \right] \end{aligned}$$

In other words, each combination

$$\left(\left[r(x_1, u) + \sum_{z} v_{u,z,1}^{k(z)} \right] \left[r(x_2, u) + \sum_{z} v_{u,z,2}^{k(z)} \right] \cdots \left[r(x_N, u) + \sum_{z} v_{u,z,N}^{k(z)} \right] \right)$$

makes for a new linear constraint in the value function V_T.

There will be one such constraint for each unique joint setting of the variables $k(1), k(2), \ldots, k(M)$. Obviously, the maximum of these linear functions is once again piecewise linear and convex, which proves that this representation indeed is sufficient to represent the correct value function over the underlying continuous belief space. Further, the number of linear pieces will be doubly exponential in the size of the measurement space, at least for our naive implementation that retains all such constraints.

15.5 Practical Considerations

The value iteration algorithm discussed thus far is far from practical. For any reasonable number of distinct states, measurements, and controls, the

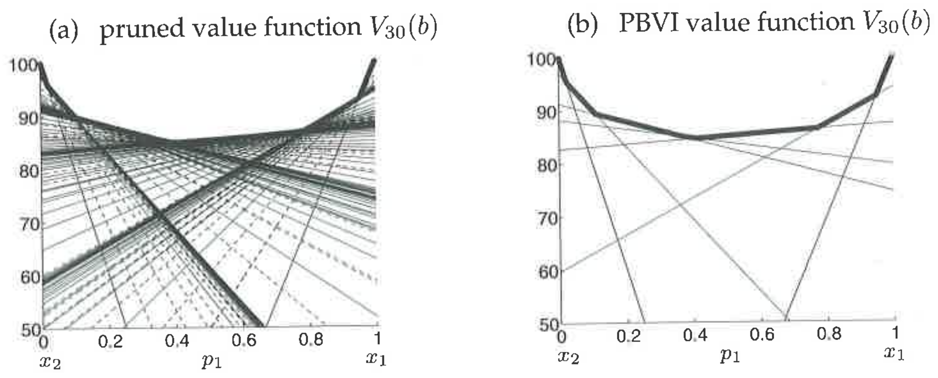


Figure 15.7 The benefit of point-based value iteration over general value iteration: Shown in (a) is the exact value function at horizon $T = 30$ for a different example, which consists of 120 constraints, after pruning. On the right is the result of the PBVI algorithm retaining only 11 linear functions. Both functions yield virtually indistinguishable results when applied to control.

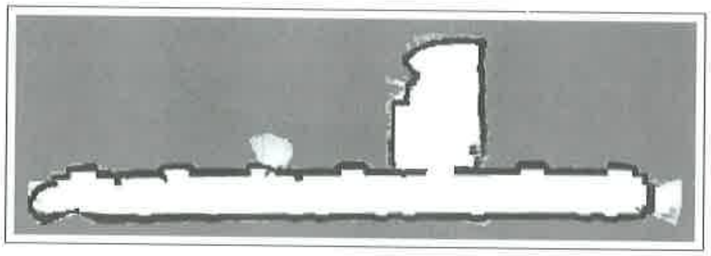
complexity of the value function is prohibitive, even for relatively beginning planning horizons.

There exists a number of opportunities to implement more efficient algorithms. One was already discussed in our example: The number of linear constraints rapidly grows prohibitively large. Fortunately, a good number of linear constraints can safely be ignored, since they do not participate in the definition of the maximum.

Another related shortcoming of the value iteration algorithm is that it computes value functions for *all* belief states, not just for the relevant ones. When a robot starts at a well-defined belief state, the set of reachable belief states is often much smaller. For example, if the robot seeks to move through two doors for which it is uncertain as to whether they are open or closed, it surely knows the state of the first when reaching the second. Thus, a belief state in which the second door's state is known but the first one is not is physically unattainable. In many domains, huge subspaces of the belief space are unattainable.

Even for the attainable beliefs, some might only be attained with small probability; others may be plainly undesirable so that the robot will generally avoid them. Value iteration makes no such distinctions. In fact, the time and resources invested in value computations are independent of the odds that a belief state will actually be relevant.

(a)



(b)

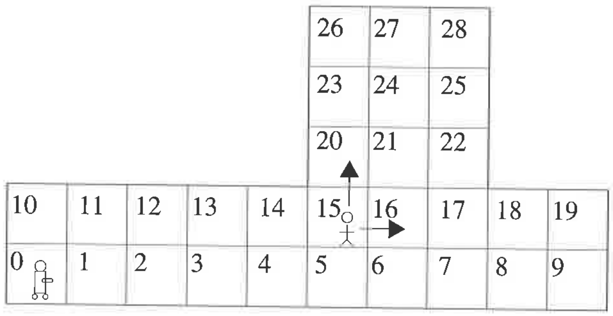


Figure 15.8 Indoor environment, in which we seek a control policy for finding a moving intruder. (a) Occupancy grid map, and (b) discrete state set used by the POMDP. The robot tracks its own pose sufficiently well that the pose uncertainty can be ignored. The remaining uncertainty pertains to the location of the person. Courtesy of Joelle Pineau, McGill University.

POINT-BASED VALUE ITERATION

There exists a flurry of algorithms that are more selective with regards to the subspace of the belief state for which a value function is computed. One of them is *point-based value iteration*, or *PBVI*. It is based on the idea of maintaining a set of exemplary belief states, and restricting the value function to constraints that maximize the value function for at least one of these belief states. More specifically, imagine we are given a set $B = \{b_1, b_2, \ldots\}$ of belief states, called *belief points*. Then the *reduced value function* V with respect to B is the set of constraints $v \in V$ for which we can find at least one $b_i \in B$ such that $v(b_i) = V(b_i)$. In other words, linear segments that do not coincide with any of the discrete belief points in B are discarded. The original PBVI algorithm calculates the value function efficiently by not even generating constraints that are not supported by any of the points; however, the same idea can also be implemented by pruning away all line segments after generating them in a standard POMDP backup.

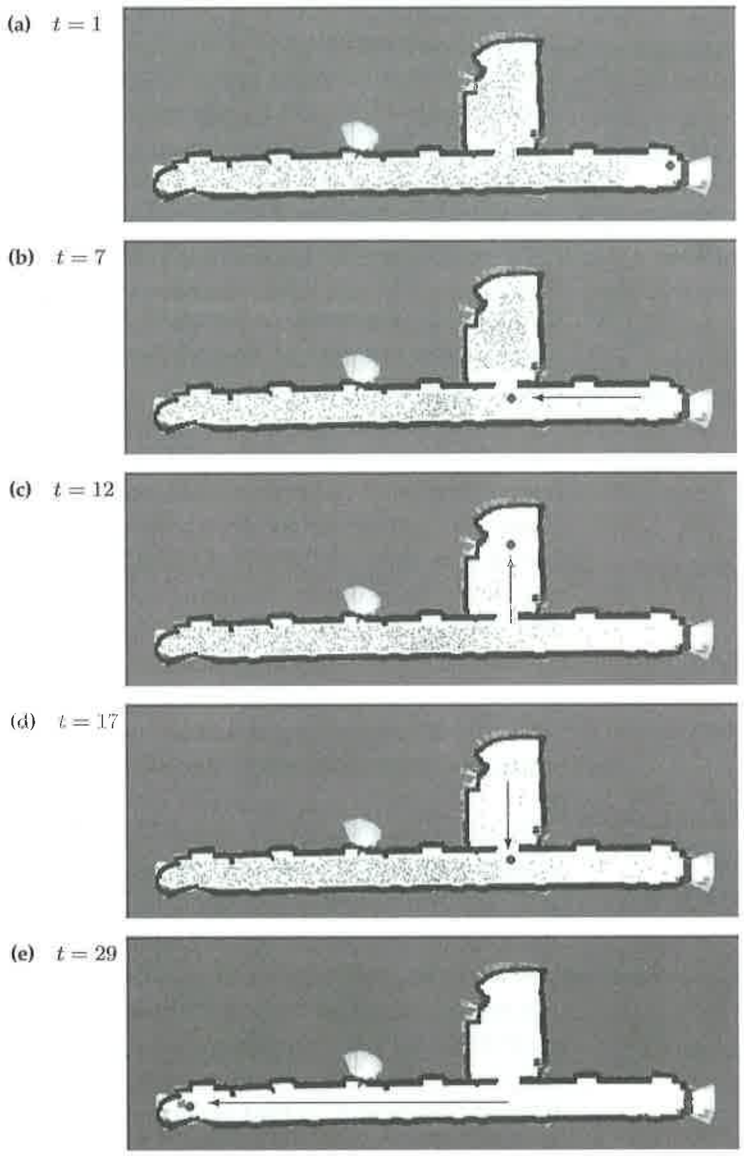


Figure 15.9 A successful search policy. Here the tracking of the intruder is implemented via a particle filter, which is then projected into a histogram representation suitable for the POMDP. The robot first clears the room on the top, then proceeds down the hallway. Courtesy of Joelle Pineau, McGill University.

The idea of maintaining a belief point set B can make value iteration significantly more efficient. Figure 15.7a shows the value function for a problem that differs from our example in Chapter 15.2 by only one aspect: The state transition function is deterministic (simply replace 0.8 by 1.0 in (15.7) and (15.8)). The value function in Figure 15.7a is optimal with respect to the horizon $T = 30$. Careful pruning along the way reduced it to 120 constraints, instead of the $10^{561,012,337}$ that a non-pruning implementation would give us—assuming the necessary patience. With a simple point set $B = \{p_1 = 0.0, p_1 = 0.1, p_1 = 0.2, \ldots, p_1 = 1\}$, we obtain the value function shown on the right side of Figure 15.7b. This value function is approximate, and it consists of only 11 linear functions. More importantly, its calculation is more than 1,000 times faster.

The use of belief points has a second important implication: The problem solver can select belief points deemed relevant for the planning process. There exists a number of heuristics to determine a set of belief points. Chief among them are to identify reachable beliefs (e.g., through simulating the robot in the POMDP), and to find beliefs that are spaced reasonably far apart from each other. By doing so it is usually possible to get many orders of magnitude faster POMDP algorithms. In fact, it is possible to grow the set B incrementally, and to therefore build up the set of value functions $V_1, v_2, \ldots, V_T$ incrementally, by adding new linear constraints to all of them whenever a new belief point is added. In this way, the planning algorithm becomes *anytime*, in that it produces increasingly better results as time goes on.

An emerging view in robotics is that the number of plausible belief states exceeds that of the number of states only by a constant factor. As a consequence, techniques that actively select appropriate regions in belief space for updating during planning have fundamentally different scaling properties than the flat, unselective value iteration approach.

A typical robotics result of PBVI is shown in Figures 15.8 and 15.9. Figure 15.8a depicts an occupancy grid map of an indoor environment that consists of a long corridor and a room. The robot starts on the right side of the diagram. Its task is to find an intruder that moves according to Brownian motion. To make this task amenable to PBVI planning, a low-dimensional state space is required. The state space used here is shown in Figure 15.8b. It tessellates the grid map into 22 discrete regions. The granularity of this representation is sufficient to solve this task, while it makes computing the PBVI value function computationally feasible. The task of finding such an intruder is inherently probabilistic. Any control policy has to be aware of the uncertainty in the environment, and seek its reduction. Further, it is inherently dynamic. Just moving to spaces not covered yet is generally insufficient. Figure 15.9 shows a typical result of POMDP planning. Here the robot has determined a control sequence that first explores the relatively small room, then progresses down the corridor. This control policy exploits the fact that while the robot clears the room, the intruder has insufficient time to pass through the corridor. Hence, this policy succeeds with high probability.

This example is paradigmatic of applying POMDP value iteration to actual robot control problem. Even when using aggressive pruning as in PBVI, the resulting value functions are still limited to a few dozen states. However, if such a low-dimensional state representation can be found, POMDP techniques yield excellent results through accommodating the inherent uncertainty in robotics.

15.6 Summary

In this section, we introduced the basic value iteration algorithm for robot control under uncertainty.

- POMDPs are characterized by multiple types of uncertainty: Uncertainty in the control effects, uncertainty in perception, and uncertainty with regards to the environment dynamics. However, POMDPs assume that we are given a probabilistic model of action and perception.

- The value function in POMDPs is defined over the space of all beliefs robots might have about the state of the world. For worlds with N states, this belief is defined over the $(N-1)$-dimensional belief simplex, characterized by the probability assigned to each of the N states.

- For finite horizons, the value function is piecewise linear in the belief space parameters. It is also continuous and convex. Thus, it can be represented as a maximum of a collection of finitely many linear functions. Further, these linear constraints are easily calculated.

- The POMDP planning algorithm computes a sequence of value functions, for increasing planning horizons. Each such calculation is recursive: Given the optimal value function at horizon $T-1$, the algorithm proceeds to computing the optimal value function at horizon T.

- Each recursive iteration combines a number of elements: The action choice is implemented by maximizing over sets of linear constraints, where each action carries its own set. The anticipated measurement is incorporated by combining sets of linear constraints, one for each measurement. The prediction is then implemented by linearly manipulating the set of linear constraints. Payoff is generalized into the belief space by calculating its expectation, which once again is linear in the belief space parameters. The result is a value backup routine that manipulates linear constraints.

- We find that the basic update produces intractably many linear constraints. Specifically, in each individual backup the measurement step increases the number of constraints by a factor that is exponential in the number of possible measurements. Most of these constraints are usually passive, and omitting them does not change the value function at all.

- Point-based value iteration (PBVI) is an approximate algorithm that maintains only constraints that are needed to support a finite set of representative belief states. In doing so, the number of constraints remains constant instead of growing doubly exponentially (in the worst case). Empirically PBVI provides good results when points are chosen to be representative and well-separated in the belief space.

In many ways, the material presented in this chapter is of theoretical interest. The value iteration algorithm defines the basic update mechanism that underlies a great number of efficient decision making algorithms. However, it in itself is not computationally tractable. Efficient implementations therefore resort to approximations, such as the PBVI technique we just discussed.

15.7 Bibliographical Remarks

EXPERIMENTAL DESIGN

The topic of decision making under uncertainty has been studied extensively in statistics, where it is known as *experimental design*. Key textbooks in this area include those by Winer et al. (1971) and Kirk and Kirk (1995); more recent work can be found in Cohn (1994).

The value iteration algorithm described in this paper goes back to Sondik (1971) and Smallwood and Sondik (1973), who were among the first to study the POMDP problem. Other early work can be found in Monahan (1982), with an early grid-based approximation in Lovejoy (1991). Finding policies for POMDPs was long deemed infeasible due to the enormous computational complexity involved. The problem was introduced into the field of Artificial Intelligence by Kaelbling et al. (1998). The pruning algorithms in Cassandra et al. (1997) and Littman et al. (1995) led to significant improvements over previous algorithms. Paired with remarkable increase of computer speed and memory available, their work enabled POMDPs to grow into a tool for solving small AI problems. Hauskrecht (1997) provided bounds on the complexity of POMDP problem solving.

The most significant wave of progress came with the advent of approximate techniques—some of which will be discussed in the next chapter. An improved grid approximation of POMDP belief spaces was devised by Hauskrecht (2000); variable resolution grids were introduced by Brafman (1997). Reachability analysis began to play a role in computing policies. Poon (2001) and Zhang and Zhang (2001) developed point-based POMDP techniques, in which the set of belief states were limited. Unlike Hauskrecht's (2000) work, these techniques relied on piecewise linear functions for representing value functions. This work culminated in the definition of the point based value iteration algorithm by Pineau et al. (2003b), who developed new any-time techniques for finding relevant belief space for solving POMDPs. Their work was later extended using tree-based representations (Pineau et al. 2003a).

Geffner and Bonet (1998) solved a number of challenge problems using dynamic programming applied to a discrete version of the belief space. This work was extended by Likhachev et al. (2004), who applied the A* algorithm (Nilsson 1982) to a restricted type of POMDP. Ferguson et al. (2004) extended this to D* planning for dynamic environments (Stentz 1995).

Another family of techniques used particles to compute policies, paired with nearest neighbor in particle set space to define approximations to the value function (Thrun 2000a). Particles were also used for POMDP monitoring by Poupart et al. (2001). Poupart and Boutilier (2000) devised an algorithm for approximating the value function using a technique sensitive to the value itself, which led to state-of-the-art results. A technique by Dearden and Boutilier (1994) gained efficiency through interleaving planning and execution of partial policies; see Smith and Simmons (2004) for additional research on interleaving heuristic search-type planning and execution. Exploiting domain knowledge was discussed in Pineau et al. (2003c), and Washington (1997) provided incremental techniques with bounds. Additional work on approximate POMDP solving is discussed in Aberdeen (2002); Murphy (2000b). One of the few fielded systems controlled by POMDP value iteration is the CMU Nursebot, whose high-level controller and dialog manager is a POMDP (Pineau et al. 2003d; Roy et al. 2000).

An alternative approach to finding POMDP control policies is to search directly in the space of policies, without computing a value function. This idea goes back to Williams (1992), who developed the idea of policy gradient search in the context of MDPs. Contemporary techniques for policy gradient search is described in Baxter et al. (2001) and Ng and Jordan (2000). Bagnell and Schneider (2001) and Ng et al. (2003) successfully applied this approach to the control of hovering an autonomous helicopter; in fact, Ng et al. (2003) reports that it took only 11 days to design such a controller using POMDP techniques, using a learned model. In more recent work, Ng et al. (2004) used these techniques to identify a controller capable of sustained inverted helicopter flight, a previously open problem. Roy and Thrun (2002) applied policy search techniques to mobile robot navigation, and discuss the combination of policy search and value iteration techniques.

Relatively little progress has been made on *learning* POMDP models. Early attempts to learn the model of a POMDP from interaction with an environment essentially failed (Lin and Mitchell 1992; Chrisman 1992), due to the hardness of the problem. Some more recent work on learning hierarchical models shows more promise (Theocharous et al. 2001). Recent work has moved away from learning HMM-style models, into alternative representations. Techniques for representing and learning the structure of partially observable stochastic environments can be found in Jaeger (2000); Littman et al. (2001); James and Singh. (2004); Rosencrantz et al. (2004). While none of these papers fully solve the POMDP problem, they nevertheless are intellectually relevant and promise new insights into the largely open problem of probabilistic robot control.

15.8 Exercises

TIGER PROBLEM

1. This problem is known as the *tiger problem* and is due to Cassandra, Littman and Kaelbling (Cassandra et al. 1994). A person faces two doors. Behind one is a tiger, behind the other a reward of $+10$. The person can either listen or open one of the doors. When opening the door with a tiger, the person will be eaten, which has an associated cost of -20. Listening costs -1. When listening, the person will hear a roaring noise that indicates the presence of the tiger, but only with 0.85 probability will the person be able to localize the noise correctly. With 0.15 probability, the noise will appear as if it came from the door hiding the reward.

   Your questions:

   (a) Provide the formal model of the POMDP, in which you define the state, action, and measurement spaces, the cost function, and the associated probability functions.

   (b) What is the expected cumulative payoff/cost of the open-loop action sequence: "Listen, listen, open door 1"? Explain your calculation.

   (c) What is the expected cumulative payoff/cost of the open-loop action sequence: "Listen, then open the door for which we did not hear a noise"? Again, explain your calculation.

   (d) Manually perform the one-step backup operation of the POMDP. Plot the resulting linear functions in a diagram just like the ones in Chapter 15.2. Provide diagrams of all intermediate steps, and don't forget to add units to your diagrams.

   (e) Manually perform the second backup, and provide all diagrams and explanations.

   (f) Implement the problem, and compute the solution for the planning horizons $T = 1, 2, \ldots, 8$. Make sure you prune the space of all linear functions. For what sequences of measurements would a person still choose to listen, even after 8 consecutive listening actions?

2. Show the correctness of Equation (15.26).

3. What is the worst-case computational complexity of a single POMDP value function backup? Provide your answer using the $O(\ )$ notation, where arguments may include the number of linear functions before a backup, and the number of states, actions, and measurements in a discrete POMDP.

4. The POMDP literature often introduces a *discount factor*, which is analogous to the discount factor discussed in the previous section. Show that even with a discount factor, the resulting value functions are still piecewise linear.

5. Consider POMDP problems with finite state, action, and measurement space, but for which the horizon $T \uparrow \infty$.

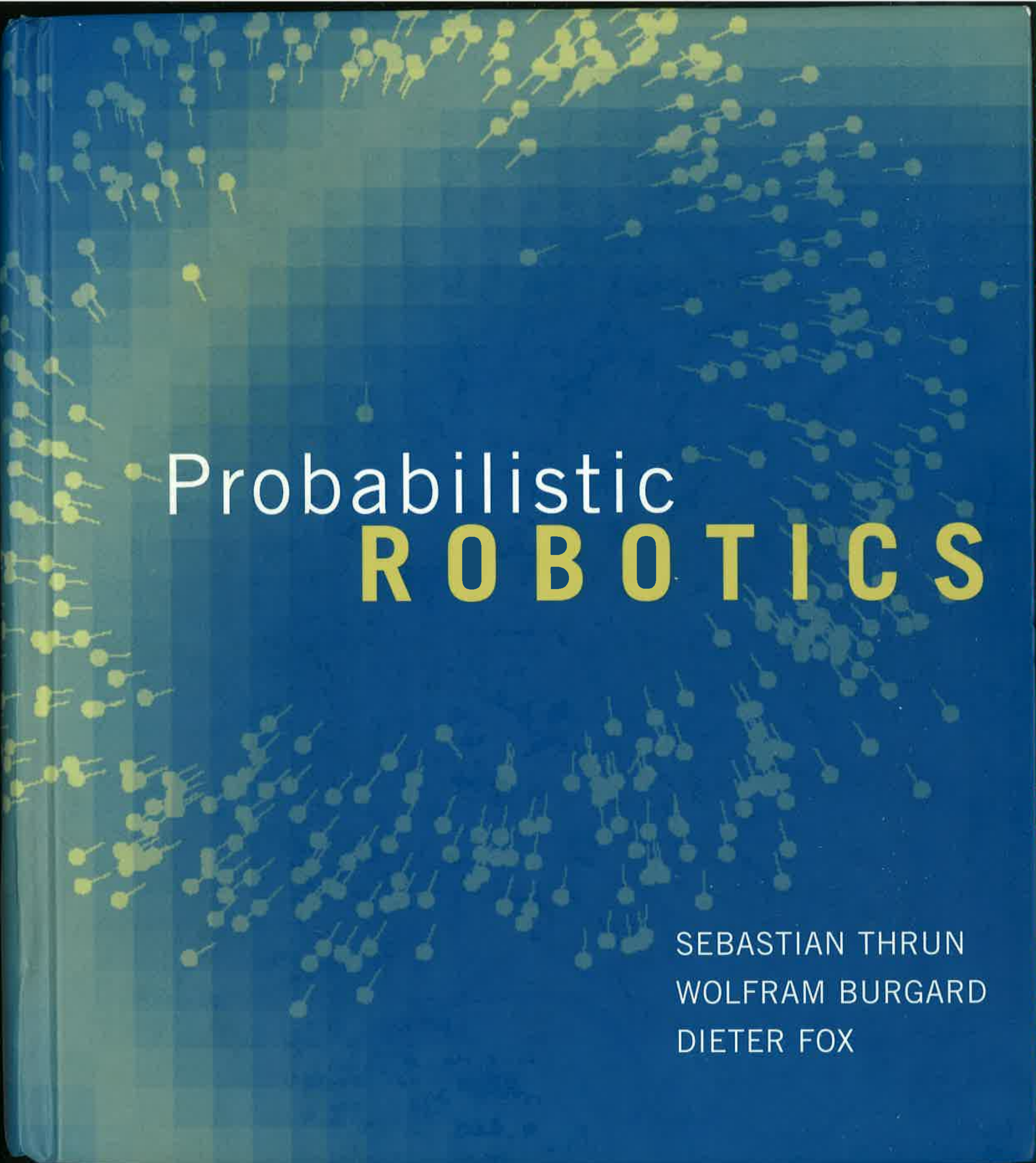
   (a) Will the value function still be piecewise linear?

   (b) Will the value function still be continuous?

   (c) Will the value function still be convex?

   For all three questions, argue why the answer is positive, or provide a counterexample in case it is negative.

6. On page 28, we provided an example of a robot sensing and opening a door. In this exercise, you are asked to implement a POMDP algorithm for an optimal control policy. Most information can be found in the example on page 28. To turn this into a control task, let us assume that the robot has a third action: **go**. When it goes, it receives $+10$ payoff if the door is open, and -100 if it is closed. The action **go** terminates the episode. The action **do_nothing** costs the robot -1, and **push** costs the robot -5. Plot value functions for different time horizons up to $T = 10$, and explain the optimal policy.



Probabilistic ROBOTICS

SEBASTIAN THRUN
WOLFRAM BURGARD
DIETER FOX