REINFORCEMENT LEARNING AND STOCHASTIC OPTIMIZATION

A unified framework for sequential decisions

Warren B. Powell

August 22, 2021

WILEY-INTERSCIENCE

A JOHN WILEY & SONS, INC., PUBLICATION

PART V - LOOKAHEAD POLICIES

Lookahead policies are based on estimates of the impact of a decision on the future. There are two broad strategies for doing this:

Value function approximations If we are in a state S_t and take an action x_t, then we observe new information W_{t+1} (which is random at time t) which takes us to a new state S_{t+1}, we might be able to approximate the value of being in state S_{t+1}. We can then use this to help us make a better decision x_t now (if we can do a good job of approximating the value of being in state S_{t+1}).

Direct lookahead approximations Here we explicitly plan decisions now, x_t, and into the future, $x_{t+1}, \ldots, x_{t+H}$, to help us make the best decision x_t to implement now. The problem in stochastic models is that the decisions $x_{tt'}$ for $t' > t$ depend on future information, so they are random.

The choice between using value functions versus direct lookaheads boils down to a single equation which gives the optimal policy at time t when we are in state S_t:

$$X_t^{\pi^*}(S_t) = \arg\max_{x_t \in \mathcal{X}_t} \left(C(S_t, x_t) + \underbrace{\mathbb{E}\left\{ \max_{\pi \in \Pi} \mathbb{E}\left\{ \sum_{t'=t+1}^{T} C(S_{t'}, X_{t'}^{\pi}(S_{t'})) | S_{t+1} \right\} | S_t, x_t \right\}}_{\text{future contributions}} \right). \tag{13.37}$$

The challenge is balancing the contributions now, given by $C(S_t, x_t)$, against future contributions. If we could compute the future contributions, this would be an optimal

policy. However, computing future contributions in the presence of a (random) sequential information process is almost always computationally intractable.

There are problems where we can create reasonable approximations of the future contributions. When we do this around the post-decision state S_t^x (we can also write this as (S_t, x_t)), this would be called the post-decision value function that we write as $\overline{V}_t^x(S_t^x|\theta)$, and allows us to write our policy as

$$X_t^{VFA}(S_t|\theta) = \arg\max_{x_t \in \mathcal{X}_t} (C(S_t, x_t) + \overline{V}_t^x(S_t^x|\theta)). \tag{13.38}$$

Needless to say, the VFA policy in equation (13.38) looks a lot friendlier than the full DLA policy using equation (13.37). The challenge is creating a reasonably accurate approximation $\overline{V}_t^x(S_t^x|\theta)$ where

$$\overline{V}_t^x(S_t^x|\theta) \approx \mathbb{E}\left\{\max_{\pi\in\Pi} \mathbb{E}\left\{\sum_{t'=t+1}^{T} C(S_{t'}, X_{t'}^{\pi}(S_{t'}))S_{t+1}\right\}|S_t, x_t\right\}.$$

This begs the question: Can we create a sufficiently accurate approximation $\overline{V}_t^x(S_t^x|\theta)$? The answer is . . . sometimes. It really depends on the problem.

Policies based on value functions have attracted considerable attention over the years from the academic community. In fact, terms like "dynamic programming" and "optimal control" are basically synonymous with value functions (or cost-to-go functions, as they are known in control theory). There are very small classes of problems where these can be computed exactly, hence the interest in fields that go by names like "approximate dynamic programming," "adaptive dynamic programming," or "reinforcement learning," although reinforcement learning has evolved to refer to an entire spectrum of policies that span, in the language of this book, all four classes of policies.

There is a wide range of strategies for approximating value functions which we have reviewed in chapter 3, all with their own strengths and weaknesses. The richness of these strategies explains why our coverage of VFA policies spans the following chapters:

Chapter 14: Exact dynamic programming - This chapter focus on a handful of sequential decision problems that can be solved exactly, which is to say, we can find provably optimal policies. Most of this presentation is centered on a field known as discrete Markov decision processes, which originated in the 1950s, and focuses on problems where there is a (not too large) set of discrete states, a (not too large) set of discrete actions, and random information W_{t+1} which allows us to take expectations. If these conditions are satisfied, these problems can be solved using a strategy that involves stepping backward through time computing the value of being in each state (this is often known as "backward dynamic programming"). The theory is very elegant, but it is rarely computable. However, the ideas lay the foundation for a variety of approximation strategies. We also touch on a special problem in optimal control called linear quadratic regulation which is a foundational result of the very large field of optimal control, with many applications in control of robots and aircraft.

Chapter 15: Backward approximate dynamic programming - This chapter describes how to do backward dynamic programming approximately for multidimensional (and even continuous) states, multidimensional (and even continuous) decisions, and complex, multidimensional exogenous information processes.

Chapter 16: Forward ADP I: The value of a policy - This chapter describes the fundamentals of approximating value functions for a fixed policy using forward methods,

where we simulate forward in time. Forward methods create a natural mechanism for sampling states (pay attention to how we sample states in chapter 15).

Chapter 17: Forward ADP II: Policy optimization - This chapter extends the previous one by showing how to simultaneously learn and optimize over policies. The interaction between learning a value function while also searching for policies introduces a significant level of complexity that explains why this field is so rich.

Chapter 18: Forward ADP III: Convex functions - This chapter adapts the forward ADP methods to the context of convex problems, specifically motivated by resource allocation problems, which represents a massive problem class. Convexity (concavity when maximizing) makes it possible for us to handle very high-dimensional problems.

By contrast, we have a single chapter, chapter 19, on direct lookahead (DLA) policies for solving equation (13.37). Our core strategy for solving equation (13.37) will be to replace the base model with an approximate lookahead model that is easier to solve, while continuing to capture the most important elements of the problem.

Our approximate lookahead model might be deterministic or stochastic. If it is deterministic, we are going to assume that algorithms are available for solving the lookahead model. If it is stochastic, then we are faced with solving a stochastic optimization within the policy for our stochastic optimization problem, albeit a simplified one. Entire fields have been dedicated to specific strategies for approximating and solving lookahead models, but these methods basically draw on all the tools of the rest of the book. For this reason, chapter 19 focuses more on strategies for creating the lookahead model, since the entire rest of the book covers the methods for solving the lookahead model.

A brief history of approximate dynamic programming

Since the 1950's the standard approach for solving sequential decision problems (dynamic programs, optimal control problems) has been to start by stating Bellman's equation (or equivalently, the Hamilton-Jacobi equation) which characterizes an optimal policy. However, almost invariably these cannot actually be computed, so the natural approach has been to solve these equations approximately. By now, as you can see from our presentation in chapter 11, and then the discussion of PFAs and CFAs in chapters 12 and 13, we feel a more balanced perspective is needed.

Approximate dynamic programming has a long history of re-invention by different communities. The first attempt was in 1959 by Bellman himself when he realized that his use of discrete states would explode when there were multiple state variables, a behavior that became widely known as the "curse of dimensionality." Computational work in the core Markov decision process community largely died at that point, with subsequent work focusing more on the theory that is summarized in chapter 14.

In 1974, Paul Werbos showed how to derive estimates of value functions for control problems using a method he called "backpropagation," which initiated a long line of research in the controls community, primarily for continuous, deterministic problems, that continues today. In fact it was this community that initiated the use of neural networks for approximating what they called "cost to go" functions (value functions in this book).

Then, in the 1980s, Rich Sutton and his adviser Andy Barto were experimenting with learning algorithms in psychology, using the setting of describing how a mouse would learn to navigate a maze. Psychology has a long history, dating to 1897 with the research by Ivan Pavlov into training dogs to associate a particular signal, or cue (in this case ringing a

bell), to elicit a response (salivating) at which point the dog would receive a treat. Through repeated trials, the dog could be trained to associate the ringing of a bell with receiving a treat that would cause the dog to salivate. The repeated trials reinforced the relationship between the bell and receiving a treat (and then salivating). This became known as "cue learning" (where the ringing of the bell is the "cue"), and the process of associating the cue with the reward became known as "reinforcement learning," terms that became popular in the 1940s and 1950s.

Sutton and Barto applied this same idea in the context of a maze, where a reward is not received until the mouse learns to find a path to a particular exit where there is a reward. As a result, an action does not immediately return a reward; instead, it just takes the mouse to a downstream state, which may eventually lead to a reward. This means the value of the action action (turning left or right) depends on the state. They designed an algorithm that would, through many repetitions, learn "Q" factors, where $Q(s,a)$ is the value of taking an action a when the mouse is in state s. The algorithm has two basic steps:

$$\hat{q}^{n+1}(s^n,a^n) = r(s^n,a^n) + \lambda \max_{a'} \bar{Q}^n(s^{n+1},a'), \tag{13.39}$$

$$\bar{Q}^{n+1}(s^n,a^n) = (1-\alpha_n)\bar{Q}^n(s^n,a^n) + \alpha_n \hat{q}^{n+1}(s^n,a^n). \tag{13.40}$$

The variables s^n and a^n is a current state and action (chosen according to rules that have to be designed). λ plays the role of a discount factor, but this has nothing to do with the time value of money. s^{n+1} is either observed from a physical system, or simulated from a known transition function given s^n and a^n. α_n is known variously as a stepsize or learning rate.

Equations (13.39) and (13.40) can be rewritten

$$\bar{Q}^{n+1}(s^n,a^n) = \bar{Q}^n(s^n,a^n) + \alpha_n(r(s^n,a^n) + \lambda \max_{a'} \bar{Q}^n(s^{n+1},a') - \bar{Q}^n(s^n,a^n)). \tag{13.41}$$

The quantity

$$(r(s^n,a^n) + \lambda \max_{a'} \bar{Q}^n(s^{n+1},a') - \bar{Q}^n(s^n,a^n))$$

became known in the reinforcement learning literature as a "temporal difference" with parameter λ, and as a result the update became known as "TD(λ)" (pronounced tee-dee-lambda). The parameter λ looks like a discount factor, but it is an *algorithmic discount factor* which has nothing to do with the time value of money (we use γ for this purpose).

At some point in the 1980s the connection between equations (13.39) and (13.40) and the field of discrete Markov decision processes was made, but it was not until 1992 that John Tsitsiklis bridged the updating equations (13.39) and (13.40) with the work on stochastic approximation methods (these are the stochastic gradient methods that we covered in chapter 5) that provided the basis for a convergence proof.

Equation (13.41) should look familiar: it is basically a stochastic gradient, with the difference that $\bar{Q}^n(s^{n+1},a')$ and $\bar{Q}^n(s^n,a^n)$ are biased estimates of the true function. Tsitsiklis extended the theory on stochastic approximation methods to handle this. This work provided the spark for the landmark book *Neuro-Dynamic Programming* by Bertsekas and Tsitsiklis in 1996 which laid the theoretical foundation for convergence theory in the entire field of VFA-based policies.

For a number of years, and to some extent still today, equations (13.39) and (13.40) are most closely associated with the term "reinforcement learning." What this community has

found, along with everyone doing research based on approximating value functions, is that value function approximations are effective on a fairly limited set of problems where using machine learning to approximate the value of being in a state produces effective policies (with reasonable effort - an often overlooked issue).

Today, the second edition of Sutton and Barto's highly popular *Reinforcement Learning: An Introduction* includes policies from all four classes of policies that we have introduced in this book. The "discovery" of strategies from the different classes of policies is a pattern that has been repeated across communities that work on sequential decision problems: fields including stochastic search, simulation-optimization, optimal control, and the multi-armed bandit community have all evolved the use of policies from the different classes of policies.

As you take the plunge into the rich set of strategies of approximating value functions, make sure that you have exhausted the simpler PFAs and CFAs, as well as the DLAs that we will cover in chapter 19. Keep in mind that sequential decision problems are ubiquitous, and that all decisions have to be made with some method. Now think about how many decisions are made by solving Bellman's equation (even approximately).

CHAPTER 14

EXACT DYNAMIC PROGRAMMING

There are very specific classes of sequential decision problems that can be solved exactly, producing optimal policies. The most general class of problems fall under the umbrella known as discrete Markov decision processes, which are characterized by a (not too large) set of discrete states $\mathcal{S}$, and a (not too large) set of discrete actions $\mathcal{A}$. We deviate from our standard notation of using x for decisions to acknowledge the long history in this field of using a for action, where a is discrete (it could be an integer, a discretized continuous variable, or a categorical quantity such as color, medical treatment or product recommendation). This is the notation that has been adopted by the reinforcement learning community.

It turns out that there is a wide range of applications with discrete actions, where the number of actions is "not too large," but the requirement that the state space is "not too large" is far more restrictive in practice. However, despite this limitation (which is severe), the study of this problem class has helped to establish the theory of sequential decision problems, and has laid the foundation for different algorithmic strategies even when the assumption of small state and action spaces does not apply.

The investigation of discrete Markov decision processes attracted a mathematically sophisticated community which has largely defined the work in this field up through the 1990s. A number of the equations in this chapter, while quite elegant (and sometimes quite sophisticated), are not computable for anything other than toy problems. This style sharply contrasts with the entire rest of the book. However, the algorithms in this chapter laid the foundation for entire classes of algorithms described in chapters 15 - 18 which scale to

much larger (and in some cases extremely large) problems. We note that the foundation for this material is laid in sections 14.1 - 14.3.

Although this chapter is primarily focused on discrete dynamic programs (discrete states and actions), we pause first in section 14.4 to demonstrate how the same equations can be used to solve certain continuous problems analytically. This section should be viewed as an exercise that illustrates the key ideas of sections 14.1 - 14.3 using a toy problem that can be solved using the same tools and concepts, but without any need for numerical computation. We then close with section 14.11 that presents the foundation of a very large field known as optimal control, where we can find optimal solutions to an important problem class known as linear quadratic regulation which has many applications in engineering.

14.1 DISCRETE DYNAMIC PROGRAMMING

To understand the power of the Markov decision process framework, it is useful to return to the idea of a decision tree, illustrated in figure 14.1. We enumerate the decisions out of each decision node (squares), and the random outcomes out of each outcome node (circles). If there are 10 possible decisions and 10 possible random outcomes, our tree is 100 times bigger after one sequence of decisions and random information. If we step forward 10 steps (10 decisions followed by random information), our tree would have 100^{10} ending nodes. And this is not even a large problem (it is easy to find problems with far larger numbers of actions and outcomes). The explosive growth in the size of the decision trees is illustrated in figure 14.1, where the number of decisions and outcomes is quite small.

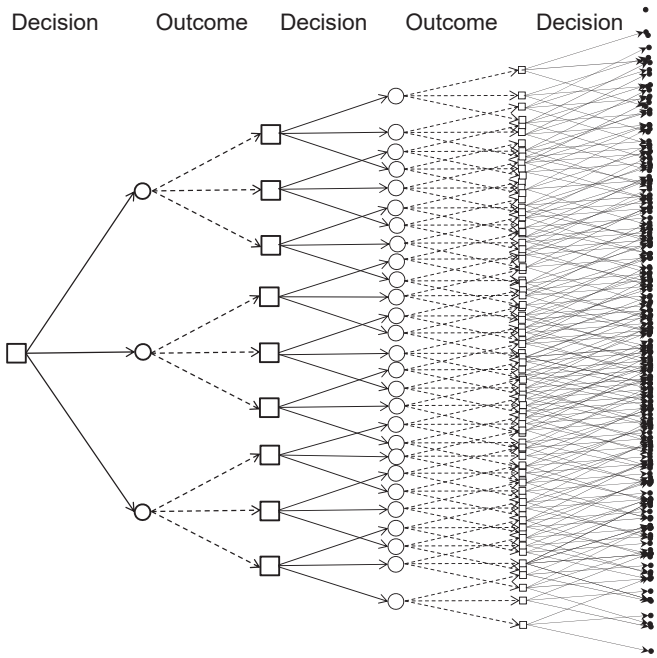


Figure 14.1 Decision tree illustrating the sequence of decisions and new information, illustrating the explosive growth of decision trees.

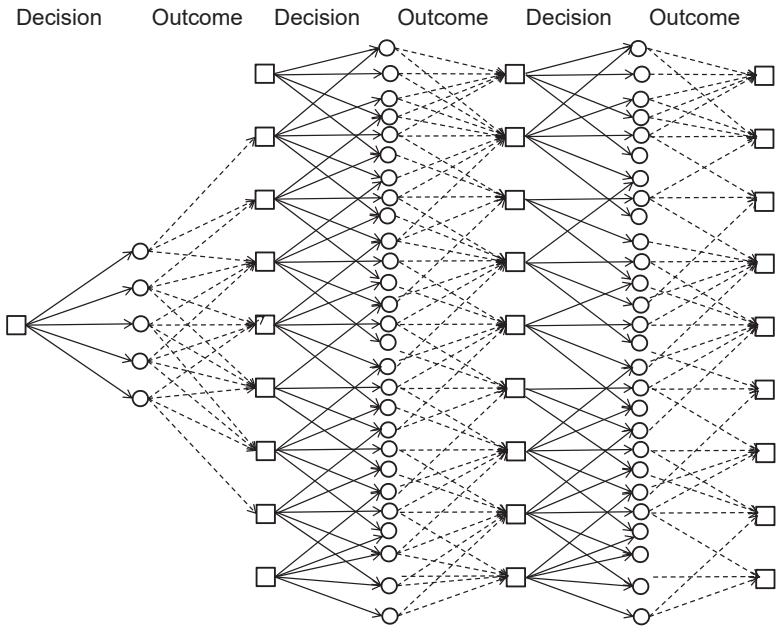


Figure 14.2 Collapsed version of the decision tree, when states do not capture entire history.

The breakthrough of Markov decision processes (by Richard Bellman in the 1950s) was the recognition that each decision node corresponds to a state of a dynamic system. In the classical representation of a decision tree, decision nodes correspond to the entire history of the process up to that point in time. However, there are many settings where we may not need the entire history to make a decision.

Assume that the relevant information we need to make a decision can be represented by a state s that falls in a discrete set $\mathcal{S} = (1, 2, \ldots, |\mathcal{S}|)$, where $\mathcal{S}$ is small enough to enumerate. For example, S_t might be the number of units of blood in a hospital inventory. In this case, the number of decision nodes does not grow exponentially. Furthermore, we only need to know the inventory, and not the history of how we got there.

When we can exploit this more compact structure, our decision tree collapses into the diagram shown in figure 14.2, where the number of states in each period is fixed. Note that the number of outcome nodes is potentially quite large (possibly infinite). For example, our random information may be continuous or multidimensional; this would be the second of the three curses of dimensionality we first introduced in section 2.1.3. (For a reminder of how complicated state variables can be, flip back to the energy storage illustrations in section 9.9.)

There are many problems where states are continuous, or the state variable is a vector producing a state space that is far too large to enumerate. In addition, the one-step transition matrix $p_t(S_{t+1}|S_t, a_t)$ can also be difficult or impossible to compute. So why cover material that is widely acknowledged to work only on small or highly specialized problems? There are (at least) four reasons:

1) Some problems have small state and action spaces and can be solved with these techniques. In fact, it is often the case that the tools of Markov decision processes offers the only path to finding the *optimal* policy.

2) We can use optimal policies, which are limited to fairly small problems, to evaluate approximation algorithms that can be scaled to larger problems.

3) The theory of Markov decision processes can be used to identify structural properties that can help us identify properties of optimal policies that we can exploit in policy search algorithms.

4) This material provides the intellectual foundation for approximation algorithms that can be scaled to far more complex problems, such as optimizing the locomotives for a major railroad, or optimizing a network of hydroelectric reservoirs.

As with most of the chapters in the book, the body of this chapter focuses on the algorithms. Some of the elegant theory that has been developed for this field is presented in the "Why does it work" section (section 14.12). The intent is to allow the presentation of results to flow more naturally, but serious students of dynamic programming are encouraged to delve into these proofs, which are quite elegant. This is partly to develop a deeper appreciation of the properties of the problem as well as to develop an understanding of the proof techniques that are used in this field.

14.2 THE OPTIMALITY EQUATIONS

In the last chapter, we illustrated a number of stochastic optimization models that involve solving the following objective function

$$\max_{\pi} \mathbb{E}\left\{\sum_{t=0}^{T} \gamma^t C(S_t, A_t^{\pi}(S_t))|S_0\right\}. \tag{14.1}$$

The most important contribution of the material in this chapter is that it provides a path to optimal policies. In practice, optimal policies are rare, so even with the computational limitations, at least having a framework for characterizing optimal policies is exceptionally valuable.

14.2.1 Bellman's equations

With a little thought, we realize that we do not have to solve this entire problem at once. Assume that we are solving a deterministic shortest path problem where S_t is the index of the node in the network where we have to make a decision. If we are in state $S_t = i$ (that is, we are at node i in our network) and take action $a_t = j$ (that is, we wish to traverse the link from i to j), our transition function will tell us that we are going to land in some state $S_{t+1} = S^M(S_t, a_t)$ (in this case, node j).

What if we had a function $V_{t+1}(S_{t+1})$ that told us the value of being in state S_{t+1} (giving us the value of the path from node j to the destination)? We could evaluate each possible action a_t and simply choose the action a_t that has the largest one-period contribution, $C_t(S_t, a_t)$, plus the value of landing in state $S_{t+1} = S^M(S_t, a_t)$ which we represent using

$V_{t+1}(S_{t+1})$. Since this value represents the money we receive one time period in the future, we might discount this by a factor γ. In other words, we have to solve

$$a_t^*(S_t) = \arg\max_{a_t \in \mathcal{A}_t} \big(C_t(S_t, a_t) + \gamma V_{t+1}(S_{t+1})\big),$$

where "$\arg\max$" means that we want to choose the action a_t that maximizes the expression in parentheses. We also note that S_{t+1} is a function of S_t and a_t, meaning that we could write it as $S_{t+1}(S_t, a_t)$. Both forms are fine. It is common to write S_{t+1} by itself, but the dependence on S_t and a_t needs to be understood.

The value of being in state S_t is the value of using the optimal decision $a_t^*(S_t)$. That is

$$\begin{aligned} V_t(S_t) &= \max_{a_t \in \mathcal{A}_t} \big(C_t(S_t, a_t) + \gamma V_{t+1}(S_{t+1}(S_t, a_t))\big) \\ &= C_t(S_t, a_t^*(S_t)) + \gamma V_{t+1}(S_{t+1}(S_t, a_t^*(S_t))). \end{aligned} \tag{14.2}$$

Equation (14.2) is the optimality equation for deterministic problems.

When we are solving stochastic problems, we have to model the fact that new information becomes available after we make the decision a_t. The result can be uncertainty in both the contribution earned, and in the determination of the next state we visit, S_{t+1}. For example, consider the problem of managing oil inventories for a refinery. Let the state S_t be the inventory in thousands of barrels of oil at time t (we require S_t to be integer). Let a_t be the amount of oil ordered at time t that will be available for use between t and $t+1$, and let $\hat{D}_{t+1}$ be the demand for oil between t and $t+1$. The state variable is governed by the simple inventory equation

$$S_{t+1}(S_t, a_t, \hat{D}_{t+1}) = \max\{0, S_t + a_t - \hat{D}_{t+1}\}.$$

We have written the state S_{t+1} using $S_{t+1}(S_t, a_t, \hat{D}_{t+1})$ to express the dependence on S_t, a_t, and $\hat{D}_{t+1}$, but it is common to simply write S_{t+1} and let the dependence on S_t, a_t and $\hat{D}_{t+1}$ be implicit. Since $\hat{D}_{t+1}$ is random at time t when we have to choose a_t, we do not know S_{t+1}. But if we know the probability distribution of the demand $\hat{D}_{t+1}$, we can work out the probability that S_{t+1} will take on a particular value.

If $\mathbb{P}^D(d) = \mathbb{P}[\hat{D} = d]$ is our probability distribution, then we can find the probability distribution for S_{t+1} using

$$Prob(S_{t+1} = s') = \begin{cases} 0 & \text{if } s' > S_t + a_t, \\ \mathbb{P}^D(S_t + a_t - s') & \text{if } 0 < s' \leq S_t + a_t, \\ \sum_{d=S_t+a_t}^{\infty} \mathbb{P}^D(d) & \text{if } s' = 0. \end{cases}$$

These probabilities depend on S_t and a_t, so we write the probability distribution as

$$\mathbb{P}(S_{t+1}|S_t, a_t) = \text{the probability of } S_{t+1} \text{ given } S_t \text{ and } a_t.$$

We can then modify the deterministic optimality equation in (14.2) by summing over the probability of each possible value of S_{t+1} (which is the same as taking an expectation), giving us

$$V_t(S_t) = \max_{a_t \in \mathcal{A}_t} \left(C_t(S_t, a_t) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(S_{t+1} = s'|S_t, a_t) V_{t+1}(s') \right). \tag{14.3}$$

We refer to this as the *standard form* of Bellman's equations, since this is the version that is used by virtually every textbook on Markov decision processes. An equivalent form is to write

$$V_t(S_t) = \max_{a_t \in \mathcal{A}_t} \big(C_t(S_t, a_t) + \gamma \mathbb{E}\{V_{t+1}(S_{t+1}(S_t, a_t, W_{t+1}))|S_t\}\big), \quad (14.4)$$

where we simply use an expectation instead of summing over probabilities. We refer to this equation as the *expectation form* of Bellman's equation. This version is the standard style that we use in this book.

Equation (14.4) is generally written in the more compact form

$$V_t(S_t) = \max_{a_t \in \mathcal{A}_t} \big(C_t(S_t, a_t) + \gamma \mathbb{E}\{V_{t+1}(S_{t+1})|S_t\}\big), \quad (14.5)$$

where the functional relationship $S_{t+1} = S^M(S_t, a_t, W_{t+1})$ is implicit. At this point, however, we have to deal with some subtleties of mathematical notation. In equation (14.4) we have captured the *functional dependence* of S_{t+1} on S_t and a_t, but the expectation is actually over the random variable W_{t+1}, which may be completely independent of the state of the system, but there may be *conditional dependence* of W_{t+1} on the state S_t and/or the action a_t. For this reason, we will often write

$$V_t(S_t) = \max_{a_t \in \mathcal{A}_t} \big(C_t(S_t, a_t) + \gamma \mathbb{E}\{V_{t+1}(S_{t+1})|S_t, a_t\}\big). \quad (14.6)$$

The standard form of Bellman's equation (14.3) has been popular in the research community since it lends itself to elegant algebraic manipulation when we assume we know the transition matrix. It is common to write it in a more compact form. Recall that a policy π is a rule that specifies the action a_t given the state S_t. In this chapter, it is easiest if we always think of a policy in terms of a rule "when we are in state s we take action a." This is a form of "lookup-table" representation of a policy that is very clumsy for most real problems, but it will serve our purposes here.

The probability that we transition from state $S_t = s$ to $S_{t+1} = s'$ can be written as

$$p_{ss'}(a) = \mathbb{P}(S_{t+1} = s'|S_t = s, a_t = a).$$

We would say that "$p_{ss'}(a)$ is the probability that we end up in state s' if we start in state s at time t when we are taking action a." Now assume that we have a function $A_t^\pi(s)$ that determines the action a we should take when in state s. It is common to write the transition probability $p_{ss'}(a)$ in the form

$$p_{ss'}^\pi = \mathbb{P}(S_{t+1} = s'|S_t = s, A_t^\pi(s) = a).$$

We can now write this in matrix form

$$P_t^\pi = \text{the one-step transition matrix under policy } \pi,$$

where $p_{ss'}^\pi$ is the element in row s and column s'. There is a different matrix P^π for each policy (decision rule) π.

Now let c_t^π be a column vector with element $c_t^\pi(s) = C_t(s, A_t^\pi(s))$, and let v_{t+1} be a column vector with element $V_{t+1}(s)$. Then (14.3) is equivalent to

$$\begin{bmatrix} \vdots \\ v_t(s) \\ \vdots \end{bmatrix} = \max_\pi \left(\begin{bmatrix} \vdots \\ c_t^\pi(s) \\ \vdots \end{bmatrix} + \gamma \begin{bmatrix} \ddots & & \\ & p_{ss'}^\pi & \\ & & \ddots \end{bmatrix} \begin{bmatrix} \vdots \\ v_{t+1}(s') \\ \vdots \end{bmatrix} \right). \quad (14.7)$$

where the maximization is performed for each element (state) in the vector. In matrix/vector form, equation (14.7) can be written

$$v_t = \max_{\pi} \left(c_t^{\pi} + \gamma P_t^{\pi} v_{t+1}\right). \tag{14.8}$$

Here, we maximize over policies because we want to find the best action for *each* state. The vector v_t is known widely as the *value function* (the value of being in each state). In control theory, it is known as the *cost-to-go function*, where it is typically denoted as J.

Equation (14.8) can be solved by finding a_t for each state s. The result is a decision vector $a_t^* = (a_t^*(s))_{s \in \mathcal{S}}$, which is equivalent to determining the best policy. This is easiest to envision when a_t is a scalar (how much to buy, whether to sell), but in many applications $a_t(s)$ is itself a vector. For example, assume our problem is to assign individual programmers to different programming tasks, where our state S_t captures the availability of programmers and the different tasks that need to be completed. Of course, computing a vector a_t for each state S_t which is itself a vector is much easier to write than to implement.

It is very easy to lose sight of the relationship between Bellman's equation and the original objective function that we stated in equation (14.1). To bring this out, we begin by writing the expected profits using policy π from time t onward

$$F_t^{\pi}(S_t) \quad = \quad \mathbb{E}\left\{\sum_{t'=t}^{T-1} C_{t'}(S_{t'}, A_{t'}^{\pi}(S_{t'})) + C_T(S_T)|S_t\right\}.$$

$F_t^{\pi}(S_t)$ is the expected total contribution if we are in state S_t in time t, and follow policy π from time t onward. If $F_t^{\pi}(S_t)$ were easy to calculate, we would probably not need dynamic programming. Instead, it seems much more natural to calculate V_t^{π} recursively using

$$V_t^{\pi}(S_t) \quad = \quad C_t(S_t, A_t^{\pi}(S_t)) + \mathbb{E}\left\{V_{t+1}^{\pi}(S_{t+1})|S_t\right\}.$$

It is not hard to show (by stepping backward in time) that

$$F_t^{\pi}(S_t) = V_t^{\pi}(S_t).$$

The proof, given in section 14.12.1, uses a proof by induction: assume it is true for V_{t+1}^{π}, and then show that it is true for V_t^{π} (not surprisingly, inductive proofs are very popular in dynamic programming).

With this result in hand, we can then establish the following key result. Let $V_t(S_t)$ be a solution to equation (14.4) (or (14.3)). Then

$$\begin{aligned} F_t^* &= \max_{\pi \in \Pi} F_t^{\pi}(S_t) \\ &= V_t(S_t). \end{aligned} \tag{14.9}$$

Equation (14.9) establishes the equivalence between (a) the value of being in state S_t and following the optimal policy and (b) the optimal value function at state S_t. While these are indeed equivalent, the equivalence is the result of a theorem (established in section 14.12.1). However, it is not unusual to find people who lose sight of the original objective function. Later, we have to solve these equations approximately, and we will need to use the original objective function to evaluate the quality of a solution.

14.2.2 Computing the transition matrix

It is very common in stochastic, dynamic programming (more precisely, Markov decision processes) to assume that the one-step transition matrix P^π is given as data (remember that there is a different matrix for each policy π). In practice, we generally can assume we know the transition function $S^M(S_t, a_t, W_{t+1})$ from which we have to derive the one-step transition matrix.

Assume that the random information W_{t+1} that arrives between t and $t+1$ is independent of all prior information. Let Ω be the set of possible outcomes of our stochastic process, and let $w_{t+1} = W_{t+1}(\omega)$ be a particular realization (for simplicity, we assume that Ω is discrete, as in a set of sampled observations), where $\mathbb{P}(W_{t+1} = w_{t+1} = W_{t+1}(\omega))$ is the probability of outcome $W_{t+1} = w_t$. Also define the indicator function

$$\mathbb{1}_{\{X\}} = \begin{cases} 1 & \text{if the statement "}X\text{" is true.} \\ 0 & \text{otherwise.} \end{cases}$$

Here, "X" represents a logical condition (such as, "is $S_t = 6$?"). We now observe that the one-step transition probability $\mathbb{P}_t(S_{t+1}|S_t, a_t)$ can be written

$$\begin{aligned} \mathbb{P}_t(S_{t+1}|S_t, a_t) &= \mathbb{E}\mathbb{1}_{\{s'=S^M(S_t,a_t,W_{t+1})\}} \\ &= \sum_{\omega\in\Omega} \mathbb{P}(W_{t+1} = w_{t+1})\mathbb{1}_{\{s'=S^M(S_t,a_t,w_{t+1})\}}. \end{aligned}$$

So, finding the one-step transition matrix means that all we have to do is to sum over all possible outcomes of the information W_{t+1} and add up the probabilities that take us from a particular state-action pair (S_t, a_t) to a particular state $S_{t+1} = s'$. Sounds easy.

In some cases, this calculation is straightforward (consider our oil inventory example earlier in the section). But in other cases, this calculation is impossible. For example, W_{t+1} might be a vector of prices or demands. In this case, the set of outcomes Ω can be much too large to enumerate (this is the third curse of dimensionality), but we can work with a sampled set of outcomes, as we indicated in section 10.3.3.

While we can estimate the transition matrix statistically, our standard approach is to simulate the transition *function*, rather than compute (or even approximate) the one step transition *matrix*. We will first see this in an ADP setting in chapter 15. For the remainder of this chapter, we assume the one-step transition matrix is available.

14.2.3 Random contributions

In many applications, the one-period contribution function is a deterministic function of S_t and a_t, and hence we routinely write the contribution as the deterministic function $C_t(S_t, a_t)$. However, this is not always the case. For example, a car traveling over a stochastic network may choose to traverse the link from node i to node j, and only learn the cost of the movement after making the decision. For such cases, the contribution function is random, and we might write it as

$\hat{C}_{t+1}(S_t, a_t, W_{t+1})$ = the contribution received in period $t+1$ given the state S_t and decision a_t, as well as the new information W_{t+1} that arrives in period $t+1$.

In this case, we simply bring the expectation in front, giving us

$$V_t(S_t) = \max_{a_t} \mathbb{E}\left\{\hat{C}_{t+1}(S_t, a_t, W_{t+1}) + \gamma V_{t+1}(S_{t+1})|S_t\right\}. \tag{14.10}$$

Now let

$$C_t(S_t, a_t) = \mathbb{E}\{\hat{C}_{t+1}(S_t, a_t, W_{t+1})|S_t\}.$$

Thus, we may view $C_t(S_t, a_t)$ as the expected contribution given that we are in state S_t and take action a_t.

14.2.4 Bellman's equation using operator notation*

The vector form of Bellman's equation in (14.8) can be written even more compactly using operator notation. Let $\mathcal{M}$ be the "max" (or "min") operator in (14.8) that can be viewed as acting on the vector v_{t+1} to produce the vector v_t. If we have a given policy π, we can write

$$\mathcal{M}^\pi v(s) = C_t(s, A^\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}_t(s'|s, A^\pi(s))v_{t+1}(s').$$

Alternatively, we can find the best action, which we represent using

$$\mathcal{M}v(s) = \max_a \big(C_t(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}_t(s'|s, a)v_{t+1}(s')\big).$$

Here, $\mathcal{M}v$ produces a vector, and $\mathcal{M}v(s)$ refers to element s of this vector. In vector form, we would write

$$\mathcal{M}v = \max_\pi \big(c_t^\pi + \gamma P_t^\pi v_{t+1}\big).$$

Now let $\mathcal{V}$ be the space of value functions. Then, $\mathcal{M}$ is a mapping

$$\mathcal{M} : \mathcal{V} \to \mathcal{V}.$$

We may also define the operator $\mathcal{M}^\pi$ for a particular policy π using

$$\mathcal{M}^\pi(v) = c_t^\pi + \gamma P^\pi v \tag{14.11}$$

for some vector $v \in \mathcal{V}$. $\mathcal{M}^\pi$ is known as a *linear operator* since the operations that it performs on v are additive and multiplicative. In mathematics, the function $c_t^\pi + \gamma P^\pi v$ is known as an *affine function*. This notation is particularly useful in mathematical proofs (see some of the proofs in section 14.12), but we will not use this notation when we describe models and algorithms.

We see later in the chapter that we can exploit the properties of this operator to derive some very elegant results for Markov decision processes. These proofs provide insights into the behavior of these systems, which can guide the design of algorithms. For this reason, it is relatively immaterial that the actual computation of these equations may be intractable for many problems; the insights still apply.

14.3 FINITE HORIZON PROBLEMS

Finite horizon problems tend to arise in two settings. First, some problems have a very specific horizon. For example, we might be interested in the value of an American option where we are allowed to sell an asset at any time $t \leq T$ where T is the exercise date.

Another problem is to determine how many seats to sell at different prices for a particular flight departing at some point in the future. In the same class are problems that require reaching some goal (but not at a particular point in time). Examples include driving to a destination, selling a house, or winning a game.

A second class of problems is actually infinite horizon, but where the goal is to determine what to do right now given a particular state of the system. For example, a transportation company might want to know what drivers should be assigned to a particular set of loads right now. Of course, these decisions need to consider the downstream impact, so models have to extend into the future, but we simply do not need to optimize over an infinite horizon. For this reason, we might model the problem over a horizon T which, when solved, yields a decision of what to do right now (this is known as a direct lookahead policy which we cover in chapter 19, but a DLA policy can involve solving a Markov decision process).

When we encounter a finite horizon problem, we assume that we are given the function $V_T(S_T)$ as data. Often, we simply use $V_T(S_T) = 0$ because we are primarily interested in what to do now, given by a_0, or in projected activities over some horizon $t = 0, 1, \ldots, H$, where H is the length of a planning horizon. If we set T sufficiently larger than H, then we may be able to assume that the decisions $a_0, a_1, \ldots, a_H$ are of sufficiently high quality to be useful.

Solving a finite horizon problem, in principle, is straightforward. The optimality equations give us

$$\begin{aligned} V_t(S_t) &= \max_{a_t \in \mathcal{A}} \mathbb{E}\left\{C_t(S_t, a_t) + \gamma V_{t+1}(S_{t+1})|S_t\right\} \\ &= \max_{a_t \in \mathcal{A}} \left(C_t(S_t, a_t) + \gamma \mathbb{E}\left\{V_{t+1}(S_{t+1})|S_t\right\}\right) \\ &= \max_{a_t \in \mathcal{A}} \left(C_t(S_t, a_t) + \gamma \sum_{s'} V_{t+1}(s') P(S_{t+1} = s'|S_t, a_t)\right), \end{aligned} \tag{14.12}$$

where $P(s'|S_t, a_t)$ is the one-step transition matrix. If we could compute the one-step transition matrix (which this community typically assumes), then all we have to do is to execute equation (14.12) starting at the last time period T (where we might assume $V_T(S_T) = 0$ or some other ending value), and then stepping backward in time (the reason why this is called "backward dynamic programming").

It is important to realize that equation (14.12) was considered a major breakthrough when it was first discovered by Richard Bellman in the 1950s. Keep in mind that prior to this work, people approached these problems as decision trees as illustrated in figure 14.1, which exploded in size extremely quickly. In effect, solving sequential stochastic optimization problems was considered completely intractable.

The implementation of backward dynamic programming is outlined in figure 14.3. The algorithm is disarmingly simple; so simple, in fact, that it is likely that this is the reason that this field has focused primarily on steady state problems, which we address below. What is overlooked, however, is that the one-step transition matrix is rarely computable, as it suffers from what is known as the three curses of dimensionality:

The state space - If the state variable S_t is an L-vector, where each dimension can take on one of K values, then the state space has K^L values, which grows very quickly with L.

The action space - The standard assumption in Markov decision processes is that a_t can take on a finite (say M, where M is not too large) values. While there are many

Step 0. Initialization:

Initialize the terminal contribution $V_T(S_T)$.

Set $t = T - 1$.

Step 1a. Step backward in time $t = T, T-1, \ldots, 0$:

Step 2a. Loop over states $s \in \mathcal{S} = \{1, \ldots, |\mathcal{S}|\}$:

Step 2b. Initialize $V_t(s) = -M$ (where M is very large).

Step 3a. Loop over each action $a \in \mathcal{A}(s)$:

Step 4a Initialize $Q(s, a) = 0$.

Step 4b. Find the expected value of being in state s and taking action a:

Step 4c. Compute $Q_t(s, a) = \sum_{w \in \mathcal{W}} \mathbb{P}(w|s, a) V_{t+1}(s' = s^M(s, a, w))$.

Step 4c. If $Q_t(s, a) > V_t(s)$ then

Step 3b. Store the best value $V_t(s) = Q_t(s, a)$.

Step 3c. Store the best action $A_t(s) = a$.

Step 1b. Return the value $V_t(s)$ and policy $A_t(s)$ for all $s \in \mathcal{S}$ and $t = 0, \ldots, T$.

Figure 14.3 A backward dynamic programming algorithm.

applications that fit this assumption, there are applications where our decision is a vector (which is the reason that we use x_t for decisions in this book), and possibly a very high-dimensional vector. Problems where x_t has ten thousand to hundred thousand dimensions arise frequently in resource allocation problems, which we illustrated in section 8.3.

The outcome space - The exogenous information W_t may also be a vector, often with continuous elements (examples of this are also found in section 8.3). The size of the outcome space grows quickly with the dimensionality of W_t.

The one-step transition matrix as $|\mathcal{S}| \times |\mathcal{S}| \times |\mathcal{A}|$ elements, and each of these elements requires an expectation over Ω. In other words, computing $P(s'|S_t, a_t)$ is the bottleneck.

We first saw backward dynamic programming in section 2.1.2 (and then again in section 14.1) when we described a simple decision tree problem. The only difference between the backward dynamic programming algorithm in figure 14.3 and our solution of the decision tree problem is primarily notational. Decision trees are visual and tend to be easier to understand, whereas in this section the methods are described using notation. However, decision tree problems are typically presented in the context of problems with relatively small numbers of states and actions: What job should I take? Should the United States put a blockade around Cuba? Should the shuttle launch have been canceled due to cold weather?

Another popular illustration of dynamic programming is the discrete asset acquisition problem. Assume that you order a quantity a_t at each time period to be used in the next time period to satisfy a demand $\hat{D}_{t+1}$. Any unused product is held over to the following time period. For this, our state variable S_t is the quantity of inventory left over at the end of the period after demands are satisfied. The transition equation is given by $S_{t+1} = [S_t + a_t - \hat{D}_{t+1}]^+$ where $[x]^+ = \max(x, 0)$. The cost function (which we seek to minimize) is given by $\hat{C}_{t+1}(S_t, a_t) = c^h S_t + c^o \mathbb{1}_{\{a_t > 0\}}$, where $\mathbb{1}_{\{X\}} = 1$ if X is true and 0 otherwise. Note that the cost function is nonconvex. This does not create problems if we

solve our minimization problem by searching over different (discrete) values of a_t. Since all of our quantities are scalar, there is no difficulty finding $C_t(S_t, a_t)$.

To compute the one-step transition matrix, let Ω be the set of possible outcomes of $\hat{D}_t$, and let $\mathbb{P}(\hat{D}_t = \omega)$ be the probability that $\hat{D}_t = \omega$. The one-step transition matrix is computed using

$$\mathbb{P}(s'|s,a) = \sum_{\omega \in \Omega} \mathbb{P}(\hat{D}_{t+1} = \omega) \mathbb{1}_{\{s'=[s+a-\omega]^+\}}$$

where Ω is the set of (discrete) outcomes of the demand $\hat{D}_{t+1}$.

Another example is the shortest path problem with random arc costs. Assume that you are trying to get from origin node q to destination node r in the shortest time possible. As you reach each intermediate node i, you are able to observe the time required to traverse each arc out of node i. Let V_j be the expected shortest path time from j to the destination node r. At node i, you see the link time $\hat{\tau}_{ij}$ which represents a random observation of the travel time. Now we choose to traverse arc (i, j^*) where j^* solves $\min_j(\hat{\tau}_{ij} + V_j)$. The choice of downstream node j^* is random since the travel time $\hat{\tau}_{ij}$ is random. We would then compute the value of being at node i using $V_i = \mathbb{E}\{\min_j(\hat{\tau}_{ij} + V_j)\}$.

14.4 CONTINUOUS PROBLEMS WITH EXACT SOLUTIONS

There is a rich history in the study of Markov decision processes of specialized problems which yield exact solutions, especially in settings with continuous states and actions. In this section we illustrate two classic problems: the gambling problem, where we derive an optimal policy for determining how much to bet, and a continuous budgeting problem. These applications nicely illustrate the core principles without hiding behind the veil of computation.

14.4.1 The gambling problem

A gambler has to determine how much of his capital he should bet on each round of a game, where he will play a total of N rounds. He will win a bet with probability p and lose with probability $q = 1 - p$ (assume $q < p$). Let S^n be his total capital after n plays, $n = 0, 1, \ldots, N$, with S^0 being his initial capital. For this problem, S^n is the state of our system (his available capital) after n plays. Let x^n be the (discrete) amount he bets in round $n + 1$, where we require that $x^n \leq S^n$. He wants to maximize $\ln S^N$, which provides a strong penalty for ending up with a small amount of money at the end and a declining marginal value for higher amounts.

Let

$$W^n = \begin{cases} 1 & \text{if the gambler wins the } n^{th} \text{ game,} \\ 0 & \text{otherwise.} \end{cases}$$

The system evolves according to

$$S^{n+1} = S^n + x^n W^{n+1} - x^n(1 - W^{n+1}).$$

Let $V^n(S^n)$ be the value of having S^n dollars at the end of the n^{th} game. The value of being in state S^n at the end of the n^{th} round can be written as

$$\begin{aligned} V^n(S^n) &= \max_{0\leq x^n\leq S^n} \mathbb{E}\{V^{n+1}(S^{n+1})|S^n\} \\ &= \max_{0\leq x^n\leq S^n} \mathbb{E}\{V^{n+1}(S^n + x^n W^{n+1} - x^n(1-W^{n+1}))|S^n\}. \end{aligned}$$

Here, we claim that the value of being in state S^n is found by choosing the decision that maximizes the expected value of being in state S^{n+1} given what we know at the end of the n^{th} round.

We solve this by starting at the end of the N^{th} trial, and assuming that we have finished with S^N dollars, which means our ending value is

$$V^N(S^N) = \ln S^N.$$

Now step back to $n = N-1$, where we may write

$$\begin{aligned} V^{N-1}(S^{N-1}) &= \max_{0\leq x^{N-1}\leq S^{N-1}} \mathbb{E}\{V^N(S^{N-1} + x^{N-1}W^N - x^{N-1}(1-W^N))|S^{N-1}\} \\ &= \max_{0\leq x^{N-1}\leq S^{N-1}} \Big[p\ln(S^{N-1}+x^{N-1}) + (1-p)\ln(S^{N-1}-x^{N-1})\Big]. \quad (14.13) \end{aligned}$$

Let $V^{N-1}(S^{N-1}, x^{N-1})$ be the value within the max operator. We can find x^{N-1} by differentiating $V^{N-1}(S^{N-1}, x^{N-1})$ with respect to x^{N-1}, giving

$$\begin{aligned} \frac{\partial V^{N-1}(S^{N-1}, x^{N-1})}{\partial x^{N-1}} &= \frac{p}{S^{N-1}+x^{N-1}} - \frac{1-p}{S^{N-1}-x^{N-1}} \\ &= \frac{2S^{N-1}p - S^{N-1} - x^{N-1}}{(S^{N-1})^2 - (x^{N-1})^2}. \end{aligned}$$

Setting this equal to zero and solving for x^{N-1} gives

$$x^{N-1} = (2p-1)S^{N-1}.$$

The next step is to plug this back into (14.13) to find $V^{N-1}(s^{N-1})$ using

$$\begin{aligned} V^{N-1}(S^{N-1}) &= p\ln(S^{N-1} + S^{N-1}(2p-1)) + (1-p)\ln(S^{N-1} - S^{N-1}(2p-1)) \\ &= p\ln(S^{N-1}2p) + (1-p)\ln(S^{N-1}2(1-p)) \\ &= p\ln S^{N-1} + (1-p)\ln S^{N-1} + \underbrace{p\ln(2p) + (1-p)\ln(2(1-p))}_{K} \\ &= \ln S^{N-1} + K, \end{aligned}$$

where K is a constant with respect to S^{N-1}. Since the additive constant does not change our decision, we may ignore it and use $V^{N-1}(S^{N-1}) = \ln S^{N-1}$ as our value function for $N-1$, which is the same as our value function for N. Not surprisingly, we can keep applying this same logic backward in time and obtain

$$V^n(S^n) = \ln S^n \ (+K^n)$$

for all n, where again, K^n is some constant that can be ignored. This means that for all n, our optimal solution is

$$x^n = (2p-1)S^n.$$

The optimal strategy at each iteration is to bet a fraction $\beta = (2p-1)$ of our current money on hand. Of course, this requires that $p > .5$.

There is a long tradition in the study of Markov decision processes of deriving the structure of optimal policies. In some cases, such as this gambling problem, we can find the optimal solution (or optimal policy). In others, we can find the structure of the policy, such as showing that a "buy low, sell high" policy is optimum, leaving us with just the problem of finding the buy and sell points.

14.4.2 The continuous budgeting problem

Assume that the resources we are allocating are continuous (for example, how much money to assign to various activities), which means that R_t is continuous, as is the decision of how much to budget. We are going to assume that the contribution from allocating x_t dollars to task t is given by

$$C_t(x_t) = \sqrt{x_t}.$$

This function assumes that there are diminishing returns from allocating additional resources to a task, as is common in many applications. We can solve this problem exactly using dynamic programming. We first note that if we have R_T dollars left for the last task, the value of being in this state is

$$V_T(R_T) = \max_{x_T \leq R_T} \sqrt{x_T}.$$

Since the contribution increases monotonically with x_T, the optimal solution is $x_T = R_T$, which means that $V_T(R_T) = \sqrt{R_T}$. Now consider the problem at time $t = T-1$. The value of being in state R_{T-1} would be

$$V_{T-1}(R_{T-1}) = \max_{x_{T-1} \leq R_{T-1}} \left(\sqrt{x_{T-1}} + V_T(R_T(x_{T-1}))\right) \tag{14.14}$$

where $R_T(x_{T-1}) = R_{T-1} - x_{T-1}$ is the money left over from time period $T-1$. Since we know $V_T(R_T)$ we can rewrite (14.14) as

$$V_{T-1}(R_{T-1}) = \max_{x_{T-1} \leq R_{T-1}} \left(\sqrt{x_{T-1}} + \sqrt{R_{T-1} - x_{T-1}}\right). \tag{14.15}$$

We solve (14.15) by differentiating with respect to x_{T-1} and setting the derivative equal to zero (we are taking advantage of the fact that we are maximizing a continuously differentiable, concave function). Let

$$F_{T-1}(R_{T-1}, x_{T-1}) = \sqrt{x_{T-1}} + \sqrt{R_{T-1} - x_{T-1}}.$$

Differentiating $F_{T-1}(R_{T-1}, x_{T-1})$ and setting this equal to zero gives

$$\begin{aligned} \frac{\partial F_{T-1}(R_{T-1}, x_{T-1})}{\partial x_{T-1}} &= \frac{1}{2}(x_{T-1})^{-\frac{1}{2}} - \frac{1}{2}(R_{T-1} - x_{T-1})^{-\frac{1}{2}} \\ &= 0. \end{aligned}$$

This implies

$$x_{T-1} = R_{T-1} - x_{T-1},$$

which gives

$$x^*_{T-1} = \frac{1}{2} R_{T-1}.$$

We now have to find V_{T-1}. Substituting x^*_{T-1} back into (14.15) gives

$$\begin{aligned} V_{T-1}(R_{T-1}) &= \sqrt{R_{T-1}/2} + \sqrt{R_{T-1}/2} \\ &= 2\sqrt{R_{T-1}/2}. \end{aligned}$$

We can continue this exercise, but there seems to be a bit of a pattern forming (this is a common trick when trying to solve dynamic programs analytically). It seems that a general formula might be

$$V_{T-t+1}(R_{T-t+1}) = t\sqrt{R_{T-t+1}/t}, \tag{14.16}$$

or, equivalently,

$$V_t(R_t) = (T-t+1)\sqrt{R_t/(T-t+1)}. \tag{14.17}$$

How do we determine if this guess is correct? We use a technique known as proof by induction. We assume that (14.16) is true for $V_{T-t+1}(R_{T-t+1})$ and then show that we get the same structure for $V_{T-t}(R_{T-t})$. Since we have already shown that it is true for V_T and V_{T-1}, this result would allow us to show that it is true for all t.

Finally, we can determine the optimal solution using the value function in equation (14.17). The optimal value of x_t is found by solving

$$\max_{x_t} \left(\sqrt{x_t} + (T-t)\sqrt{(R_t - x_t)/(T-t)} \right). \tag{14.18}$$

Differentiating and setting the result equal to zero gives

$$\frac{1}{2}(x_t)^{-\frac{1}{2}} - \frac{1}{2}\left(\frac{R_t - x_t}{T-t}\right)^{-\frac{1}{2}} = 0.$$

This implies that

$$x_t = (R_t - x_t)/(T-t).$$

Solving for x_t gives

$$x^*_t = R_t/(T-t+1).$$

This gives us the very intuitive result that we want to evenly divide the available budget among all remaining tasks. This is what we would expect since all the tasks produce the same contribution.

14.5 INFINITE HORIZON PROBLEMS*

Most of this book focuses on finite horizon problems, which tends to be most useful for practical problems. The history of research in Markov decision processes has been to focus on infinite horizon problems. We speculate that if you assume that you are given

the one-step transition matrix $P(S_{t+1} = s'|S_t = s, a)$, solving finite horizon problems become, well, trivial. Needless to say, this is far from the truth.

By contrast, infinite horizon problems are challenging with genuinely elegant mathematics, as we will see. We typically use infinite horizon formulations whenever we wish to study a problem where the parameters of the contribution function, transition function and the process governing the exogenous information process do not vary over time. More importantly, infinite horizon problems provide a number of insights into the properties of problems and algorithms, drawing off an elegant theory that has evolved around this problem class. Even students who wish to solve complex, nonstationary problems will benefit from an understanding of this problem class.

We start with the finite-horizon version of Bellman's equation, which we saw earlier but repeat here

$$V_t(S_t) = \max_{a_t \in \mathcal{A}} \mathbb{E}\left\{C_t(S_t, a_t) + \gamma V_{t+1}(S_{t+1})|S_t\right\}. \tag{14.19}$$

We can think of a steady-state problem as one without the time dimension. Letting $V(s) = \lim_{t\to\infty} V_t(S_t)$ (and assuming the limit exists), we obtain the steady-state optimality equations

$$V(s) = \max_{a\in\mathcal{A}} \left\{C(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathbb{P}(s'|s,a)V(s')\right\}. \tag{14.20}$$

The functions $V(s)$ can be shown (as we do later) to be equivalent to solving the infinite horizon problem

$$\max_{\pi\in\Pi} \mathbb{E}\left\{\sum_{t=0}^{\infty} \gamma^t C_t(S_t, A_t^\pi(S_t))\right\}. \tag{14.21}$$

Now define

$$\begin{aligned} P^{\pi,t} &= t\text{-step transition matrix, over periods } 0, 1, \ldots, t-1, \text{ given policy } \pi \\ &= \Pi_{t'=0}^{t-1} P_{t'}^\pi. \end{aligned} \tag{14.22}$$

We further define $P^{\pi,0}$ to be the identity matrix. As before, let c_t^π be the column vector of the expected cost of being in each state given that we choose the action a_t described by policy π, where the element for state s is $c_t^\pi(s) = C_t(s, A^\pi(s))$. The infinite horizon, discounted value of a policy π starting at time t is given by

$$v_t^\pi = \sum_{t'=t}^{\infty} \gamma^{t'-t} P^{\pi,t'-t} c_{t'}^\pi. \tag{14.23}$$

Assume that after following policy π_0 we follow policy $\pi_1 = \pi_2 = \ldots = \pi$. In this case, equation (14.23) can now be written as (starting at $t = 0$)

$$v^{\pi_0} = c^{\pi_0} + \sum_{t'=1}^{\infty} \gamma^{t'} P^{\pi,t'} c_{t'}^\pi \tag{14.24}$$

$$= c^{\pi_0} + \sum_{t'=1}^{\infty} \gamma^{t'} \left(\Pi_{t''=0}^{t'-1} P_{t''}^\pi\right) c_{t'}^\pi \tag{14.25}$$

$$= c^{\pi_0} + \gamma P^{\pi_0} \sum_{t'=1}^{\infty} \gamma^{t'-1} \left(\Pi_{t''=1}^{t'-1} P_{t''}^\pi\right) c_{t'}^\pi \tag{14.26}$$

$$= c^{\pi_0} + \gamma P^{\pi_0} v^\pi. \tag{14.27}$$

Equation (14.27) shows us that the value of a policy is the single period reward plus a discounted final reward that is the same as the value of a policy starting at time 1. If our decision rule is stationary, then $\pi_0 = \pi_1 = \ldots = \pi_t = \pi$, which allows us to rewrite (14.27) as

$$v^\pi = c^\pi + \gamma P^\pi v^\pi. \tag{14.28}$$

This allows us to solve for the stationary reward explicitly (as long as $0 \leq \gamma < 1$), giving us

$$v^\pi = (I - \gamma P^\pi)^{-1} c^\pi.$$

We can also write an infinite horizon version of the optimality equations using our operator notation. Letting $\mathcal{M}$ be the "max" (or "min") operator (also known as the Bellman operator), the infinite horizon version of equation (14.11) would be written

$$\mathcal{M}^\pi(v) = c^\pi + \gamma P^\pi v. \tag{14.29}$$

There are several algorithmic strategies for solving infinite horizon problems. The first, value iteration, is the most widely used method. It involves iteratively estimating the value function. At each iteration, the estimate of the value function determines which decisions we will make and as a result defines a policy. The second strategy is *policy iteration*. At every iteration, we define a policy (literally, the rule for determining decisions) and then determine the value function for that policy.

Careful examination of value and policy iteration reveals that these are closely related strategies that can be viewed as special cases of a general strategy that uses value and policy iteration. Finally, the third major algorithmic strategy exploits the observation that the value function can be viewed as the solution to a specially structured linear programming problem.

14.6 VALUE ITERATION FOR INFINITE HORIZON PROBLEMS*

Value iteration is perhaps the most widely used algorithm in dynamic programming for infinite horizon problems because it is the simplest to implement and, as a result, often tends to be the most natural way of solving many problems. It is virtually identical to backward dynamic programming for finite horizon problems. In addition, most of our work in approximate dynamic programming is based on value iteration.

Value iteration comes in several flavors. The basic version of the value iteration algorithm is given in figure 14.4. The proof of convergence (see section 14.12.2) is quite elegant for students who enjoy mathematics. The algorithm also has several nice properties that we explore below.

It is easy to see that the value iteration algorithm is similar to the backward dynamic programming algorithm. Rather than using a subscript t, which we decrement from T back to 0, we use an iteration counter n that starts at 0 and increases until we satisfy a convergence criterion. Here, we stop the algorithm when

$$\|v^n - v^{n-1}\| < \epsilon(1-\gamma)/2\gamma,$$

where $\|v\|$ is the max-norm defined by

$$\|v\| = \max_s |v(s)|.$$

Step 0. Initialization:

Set $v^0(s) = 0 \ \forall s \in \mathcal{S}$.

Fix a tolerance parameter $\epsilon > 0$.

Set $n = 1$.

Step 1. For each $s \in \mathcal{S}$ compute:

$$v^n(s) = \max_{a \in \mathcal{A}} \big(C(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s,a) v^{n-1}(s')\big). \tag{14.30}$$

Step 2. If $\|v^n - v^{n-1}\| < \epsilon(1-\gamma)/2\gamma$, let π^ϵ be the resulting policy that solves (14.30), and let $v^\epsilon = v^n$ and stop; else set $n = n + 1$ and go to step 1.

Figure 14.4 The value iteration algorithm for infinite horizon optimization

Thus, $\|v\|$ is the largest absolute value of a vector of elements. Thus, we stop if the largest change in the value of being in any state is less than $\epsilon(1-\gamma)/2\gamma$ where ϵ is a specified error tolerance.

Below, we describe a Gauss-Seidel variant which is a useful method for accelerating value iteration, and a version known as relative value iteration.

14.6.1 A Gauss-Seidel variation

A slight variant of the value iteration algorithm provides a faster rate of convergence. In this version (typically called the Gauss-Seidel variant), we take advantage of the fact that when we are computing the expectation of the value of the future, we have to loop over all the states s' to compute $\sum_{s'} \mathbb{P}(s'|s,a)v^n(s')$. For a particular state s, we would have already computed $v^{n+1}(\hat{s})$ for $\hat{s} = 1, 2, \ldots, s-1$. By simply replacing $v^n(\hat{s})$ with $v^{n+1}(\hat{s})$ for the states we have already visited, we obtain an algorithm that typically exhibits a noticeably faster rate of convergence. The algorithm requires a change to step 1 of the value iteration, as shown in figure 14.5.

Replace Step 1 with

Step 1'. For each $s \in \mathcal{S}$ compute

$$v^n(s) = \max_{a \in \mathcal{A}} \left\{ C(s,a) + \gamma \left(\sum_{s'<s} \mathbb{P}(s'|s,a) v^n(s') + \sum_{s' \geq s} \mathbb{P}(s'|s,a) v^{n-1}(s') \right) \right\}$$

Figure 14.5 The Gauss-Seidel variation of value iteration.

14.6.2 Relative value iteration

Another version of value iteration is called *relative value iteration*, which is useful in problems that do not have a discount factor or where the optimal policy converges much more quickly than the value function, which may grow steadily for many iterations. The relative value iteration algorithm is shown in 14.6.

Step 0. Initialization:

- Choose some $v^0 \in \mathcal{V}$.
- Choose a base state s^* and a tolerance ϵ.
- Let $w^0 = v^0 - v^0(s^*)e$ where e is a vector of ones.
- Set $n = 1$.

Step 1. Set

$$\begin{aligned} v^n &= \mathcal{M}w^{n-1}, \\ w^n &= v^n - v^n(s^*)e. \end{aligned}$$

Step 2. If $sp(v^n - v^{n-1}) < (1-\gamma)\epsilon/\gamma$, go to step 3; otherwise, go to step 1.

Step 3. Set $a^\epsilon = \arg\max_{a\in\mathcal{A}} \left(C(a) + \gamma P^\pi v^n\right)$.

Figure 14.6 Relative value iteration.

In relative value iteration, we focus on the fact that we may be more interested in the convergence of the difference $|v(s) - v(s')|$ than we are in the values of $v(s)$ and $v(s')$. This would be the case if we are interested in the best policy rather than the value function itself (this is not always the case). What often happens is that, especially toward the limit, all the values $v(s)$ start increasing by the same rate. For this reason, we can pick any state (denoted s^* in the algorithm) and subtract its value from all the other states.

To provide a bit of formalism for our algorithm, we define the *span* of a vector v as follows:

$$sp(v) = \max_{s\in\mathcal{S}} v(s) - \min_{s\in\mathcal{S}} v(s).$$

Note that our use of "span" is different than the way it is normally used in linear algebra. Here and throughout this section, we define the norm of a vector as

$$\|v\| = \max_{s\in\mathcal{S}} v(s).$$

Note that the span has the following six properties:

1) $sp(v) \geq 0$.
2) $sp(u+v) \leq sp(u) + sp(v)$.
3) $sp(kv) = |k|sp(v)$.
4) $sp(v+ke) = sp(v)$.
5) $sp(v) = sp(-v)$.
6) $sp(v) \leq 2\|v\|$.

Property (4) implies that $sp(v) = 0$ does not mean that $v = 0$ and therefore it does not satisfy the properties of a norm. For this reason, it is called a *semi-norm*.

The relative value iteration algorithm is simply subtracting a constant from the value vector at each iteration. Obviously, this does not change the optimal decision, but it does change the value itself. If we are only interested in the optimal policy, relative value iteration often offers much faster convergence, but it may not yield accurate estimates of the value of being in each state.

14.6.3 Bounds and rates of convergence

One important property of value iteration algorithms is that if our initial estimate is too low, the algorithm will rise to the correct value from below. Similarly, if our initial estimate is too high, the algorithm will approach the correct value from above. This property is formalized in the following theorem:

Theorem 14.6.1. *For a vector $v \in \mathcal{V}$:*

(a) If v satisfies $v \geq \mathcal{M}v$, then $v \geq v^*$.

(b) If v satisfies $v \leq \mathcal{M}v$, then $v \leq v^*$.

(c) If v satisfies $v = \mathcal{M}v$, then v is the unique solution to this system of equations and $v = v^*$.

The proof is given in section 14.12.3. It is a nice property because it provides some valuable information on the nature of the convergence path. In practice, we generally do not know the true value function, which makes it hard to know if we are starting from above or below (although some problems have natural bounds, such as nonnegativity).

The proof of the monotonicity property above also provides us with a nice corollary. If $V(s) = \mathcal{M}V(s)$ for all s, then $V(s)$ is the unique solution to this system of equations, which must also be the optimal solution.

This result raises the question: What if some of our estimates of the value of being in some states are too high, while others are too low? This means the values may cycle above and below the optimal solution, although at some point we may find that all the values have increased (decreased) from one iteration to the next. If this happens, then it means that the values are all equal to or below (above) the limiting value.

Value iteration also provides a nice bound on the quality of the solution. Recall that when we use the value iteration algorithm, we stop when

$$\|v^{n+1} - v^n\| < \epsilon(1-\gamma)/2\gamma \tag{14.31}$$

where γ is our discount factor and ϵ is a specified error tolerance. It is possible that we have found the optimal policy when we stop, but it is very unlikely that we have found the optimal value functions. We can, however, provide a bound on the gap between the solution v^n and the optimal values v^* by using the following theorem:

Theorem 14.6.2. *If we apply the value iteration algorithm with stopping parameter ϵ and the algorithm terminates at iteration n with value function v^{n+1}, then*

$$\|v^{n+1} - v^*\| \leq \epsilon/2. \tag{14.32}$$

Let π^ϵ be the policy that we terminate with, and let v^{π^ϵ} be the value of this policy. Then

$$\|v^{\pi^\epsilon} - v^*\| \leq \epsilon.$$

The proof is given in section 14.12.4. While it is nice that we can bound the error, the bad news is that the bound can be quite poor. More important is what the bound teaches us about the role of the discount factor.

We can provide some additional insights into the bound, as well as the rate of convergence, by considering a trivial dynamic program. In this problem, we receive a constant

reward c at every iteration. There are no decisions, and there is no randomness. The value of this "game" is quickly seen to be

$$\begin{aligned} v^* &= \sum_{n=0}^{\infty} \gamma^n c \\ &= \frac{1}{1-\gamma} c. \end{aligned} \tag{14.33}$$

Consider what happens when we solve this problem using value iteration. Starting with $v^0 = 0$, we would use the iteration

$$v^n = c + \gamma v^{n-1}.$$

After we have repeated this n times, we have

$$\begin{aligned} v^n &= \sum_{m=0}^{n-1} \gamma^n c \\ &= \frac{1-\gamma^n}{1-\gamma} c. \end{aligned} \tag{14.34}$$

Comparing equations (14.33) and (14.34), we see that

$$v^n - v^* = -\frac{\gamma^n}{1-\gamma} c. \tag{14.35}$$

Similarly, the change in the value from one iteration to the next is given by

$$\begin{aligned} \|v^{n+1} - v^n\| &= \left| \frac{\gamma^{n+1}}{1-\gamma} - \frac{\gamma^n}{1-\gamma} \right| c \\ &= \gamma^n \left| \frac{\gamma}{1-\gamma} - \frac{1}{1-\gamma} \right| c \\ &= \gamma^n \left| \frac{\gamma - 1}{1-\gamma} \right| c \\ &= \gamma^n c. \end{aligned}$$

If we stop at iteration $n+1$, then it means that

$$\gamma^n c \leq \epsilon/2 \left(\frac{1-\gamma}{\gamma} \right). \tag{14.36}$$

If we choose ϵ so that (14.36) holds with equality, then our error bound (from 14.32) is

$$\begin{aligned} \|v^{n+1} - v^*\| &\leq \epsilon/2 \\ &= \frac{\gamma^{n+1}}{1-\gamma} c. \end{aligned}$$

From (14.35), we know that the distance to the optimal solution is

$$|v^{n+1} - v^*| = \frac{\gamma^{n+1}}{1-\gamma} c,$$

which matches our bound.

This little exercise confirms that our bound on the error may be tight. It also shows that the error decreases geometrically at a rate determined by the discount factor. For this problem, the error arises because we are approximating an infinite sum with a finite one. For more realistic dynamic programs, we also have the effect of trying to find the optimal policy. When the values are close enough that we have, in fact, found the optimal policy, then we have only a Markov reward process (a Markov chain where we earn rewards for each transition). Once our Markov reward process has reached steady state, it will behave just like the simple problem we have just solved, where c is the expected reward from each transition.

14.7 POLICY ITERATION FOR INFINITE HORIZON PROBLEMS*

In policy iteration, we choose a policy and then find the infinite horizon, discounted value of the policy. This value is then used to choose a new policy. The general algorithm is described in figure 14.7. Policy iteration is popular for infinite horizon problems because of the ease with which we can find the value of a policy. As we showed in section 14.5, the value of following policy π is given by

$$v^{\pi} = (I - \gamma P^{\pi})^{-1} c^{\pi}. \tag{14.37}$$

While computing the inverse can be problematic as the state space grows, it is, at a minimum, a very convenient formula.

It is useful to illustrate the policy iteration algorithm in different settings. In the first, consider a batch replenishment problem where we have to replenish resources (raising capital, exploring for oil to expand known reserves, hiring people) where there are economies from ordering larger quantities. We might use a simple policy where if our level of resources $R_t < q$ for some lower limit q, we order a quantity $a_t = Q - R_t$. This policy is parameterized by (q, Q) and is written

$$A^{\pi}(R_t) = \begin{cases} 0, & R_t \geq q, \\ Q - R_t, & R_t < q. \end{cases} \tag{14.38}$$

For a given set of parameters $\pi = (q, Q)$, we can compute a one-step transition matrix P^{π} and a contribution vector c^{π}.

Policies come in many forms. For the moment, we simply view a policy as a rule that tells us what decision to make when we are in a particular state. In later chapters, we introduce policies in different forms since they create different challenges for finding the best policy.

Given a transition matrix P^{π} and contribution vector c^{π}, we can use equation (14.37) to find v^{π}, where $v^{\pi}(s)$ is the discounted value of started in state s and following policy π. From this vector, we can infer a new policy by solving

$$a^n(s) = \arg\max_{a \in \mathcal{A}} \left(C(a) + \gamma P^{\pi} v^n \right) \tag{14.39}$$

for each state s. For our batch replenishment example, it turns out that we can show that $a^n(s)$ will have the same structure as that shown in (14.38). So, we can either store $a^n(s)$ for each s, or simply determine the parameters (q, Q) that correspond to the decisions produced by (14.39). The complete policy iteration algorithm is described in figure 14.7.

Step 0. Initialization:

Step 0a. Select a policy π^0.

Step 0b. Set $n = 1$.

Step 1. Given a policy π^{n-1}:

Step 1a. Compute the one-step transition matrix $P^{\pi^{n-1}}$.

Step 1b Compute the contribution vector $c^{\pi^{n-1}}$ where the element for state s is given by $c^{\pi^{n-1}}(s) = C(s, A^{\pi^{n-1}})$.

Step 2. Let $v^{\pi,n}$ be the solution to

$$(I - \gamma P^{\pi^{n-1}})v = c^{\pi^{n-1}}.$$

Step 3. Find a policy π^n defined by

$$a^n(s) = \arg\max_{a \in \mathcal{A}} \big(C(a) + \gamma P^{\pi} v^n\big).$$

This requires that we compute an action for each state s.

Step 4. If $a^n(s) = a^{n-1}(s)$ for all states s, then set $a^* = a^n$; otherwise, set $n = n + 1$ and go to step 1.

Figure 14.7 Policy iteration

The policy iteration algorithm is simple to implement and has fast convergence when measured in terms of the number of iterations. However, solving equation (14.37) is quite hard if the number of states is large. If the state space is small, we can use $v^{\pi} = (I - \gamma P^{\pi})^{-1} c^{\pi}$, but the matrix inversion can be computationally expensive. For this reason, we may use a hybrid algorithm that combines the features of policy iteration and value iteration.

14.8 HYBRID VALUE-POLICY ITERATION*

Value iteration is basically an algorithm that updates the value at each iteration and then determines a new policy given the new estimate of the value function. At any iteration, the value function is not the true, steady-state value of the policy. By contrast, policy iteration picks a policy and then determines the true, steady-state value of being in each state given the policy. Given this value, a new policy is chosen.

It is perhaps not surprising that policy iteration converges faster in terms of the number of iterations because it is doing a lot more work in each iteration (determining the true, steady-state value of being in each state under a policy). Value iteration is much faster per iteration, but it is determining a policy given an approximation of a value function and then performing a very simple updating of the value function, which may be far from the true value function.

A hybrid strategy that combines features of both methods is to perform a somewhat more complete update of the value function before performing an update of the policy. Figure 14.8 outlines the procedure where the steady-state evaluation of the value function in equation (14.37) is replaced with a much easier iterative procedure (step 2 in figure 14.8). This step is run for M iterations, where M is a user-controlled parameter that allows the exploration of the value of a better estimate of the value function. Not surprisingly, it will

Step 0. Initialization:

- Set $n = 1$.
- Select a tolerance parameter ϵ and inner iteration limit M.
- Select some $v^0 \in \mathcal{V}$.

Step 1. Find a decision $a^n(s)$ for each s that satisfies

$$a^n(s) \quad = \quad \arg\max_{a \in \mathcal{A}} \left\{ C(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s,a) v^{n-1}(s') \right\},$$

which we represent as policy π^n.

Step 2. Partial policy evaluation.

(a) Set $m = 0$ and let: $u^n(0) = c^\pi + \gamma P^{\pi^n} v^{n-1}$.

(b) If $\|u^n(0) - v^{n-1}\| < \epsilon(1-\gamma)/2\gamma$, go to step 3. Else:

(c) While $m < M$ do the following:

i) $u^n(m+1) = c^{\pi^n} + \gamma P^{\pi^n} u^n(m) = \mathcal{M}^\pi u^n(m)$.

ii) Set $m = m + 1$ and repeat (i).

(d) Set $v^n = u^n(M), n = n + 1$ and return to step 1.

Step 3. Set $a^\epsilon = a^{n+1}$ and stop.

Figure 14.8 Hybrid value/policy iteration

generally be the case that M should decline with the number of iterations as the overall process converges.

14.9 AVERAGE REWARD DYNAMIC PROGRAMMING*

There are settings where the natural objective function is to maximize the *average* contribution per unit time. Assume we start in state s. Then, the average reward from starting in state s and following policy π is given by

$$\max_\pi F^\pi(s) = \lim_{T \to \infty} \frac{1}{T} \mathbb{E} \sum_{t=0}^{T} C(S_t, A^\pi(S_t)). \tag{14.40}$$

Here, $F^\pi(s)$ is the expected reward *per time period*. In matrix form, the total value of following a policy π over a horizon T can be written as

$$V_T^\pi = \sum_{t=0}^{T} (P^\pi)^t c^\pi,$$

where V_T^π is a column vector with element $V_T^\pi(s)$ giving the expected contribution over T time periods when starting in state s. We can get a sense of how $V_T^\pi(s)$ behaves by watching what happens as T becomes large. Assuming that our underlying Markov chain is ergodic (which means you can eventually get from any state to any other state with positive probability), we know that $(P^\pi)^T \to P^*$ where the rows of P^* are all the same.

Now define a column vector g given by

$$g^\pi = P^* c^\pi.$$

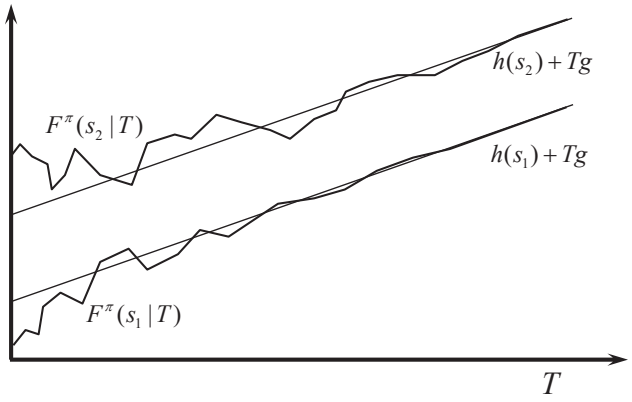


Figure 14.9 Cumulative contribution over a horizon T when starting in states s_1 and s_2, showing growth approaching a rate that is independent of the starting state.

Since the rows of P^* are all the same, all the elements of g^π are the same, and each element gives the average contribution per time period using the steady state probability of being in each state. For finite T, each element of the column vector V_T^π is not the same, since the contributions we earn in the first few time periods depends on our starting state. But it is not hard to see that as T grows large, we can write

$$V_T^\pi \rightarrow h^\pi + Tg^\pi,$$

where the vector h^π captures the state-dependent differences in the total contribution, while g^π is the state-independent average contribution in the limit. Figure 14.9 illustrates the growth in V_T^π toward a linear function.

If we wish to find the policy that performs the best as $T \rightarrow \infty$, then clearly the contribution of h^π vanishes, and we want to focus on maximizing g^π, which we can now treat as a scalar.

14.10 THE LINEAR PROGRAMMING METHOD FOR DYNAMIC PROGRAMS**

Theorem 14.6.1 showed us that if

$$v \geq \max_a \big(C(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s,a)v(s')\big),$$

then v is an upper bound (actually, a vector of upper bounds) on the value of being in each state. This means that the optimal solution, which satisfies $v^* = c + \gamma P v^*$, is the smallest value of v that satisfies this inequality. We can use this insight to formulate the problem of finding the optimal values as a linear program. Let β be a vector with elements $\beta_s > 0,\ \forall s \in \mathcal{S}$. The optimal value function can be found by solving the following linear program

$$\min_v \sum_{s \in \mathcal{S}} \beta_s v(s) \tag{14.41}$$

subject to

$$v(s) \quad \geq \quad C(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s,a) v(s') \quad \text{for all } s \text{ and } a. \tag{14.42}$$

The linear program has a $|\mathcal{S}|$-dimensional decision vector (the value of being in each state), with $|\mathcal{S}| \times |\mathcal{A}|$ inequality constraints (equation (14.42)).

This formulation was viewed as primarily a theoretical result for many years, since it requires formulating a linear program where the number of constraints is equal to the number of states times the number of actions. While even today this limits the size of problems it can solve, modern linear programming solvers can handle problems with tens of thousands of constraints without difficulty. This size is greatly expanded with the use of specialized algorithmic strategies which are an active area of research as of this writing.

The advantage of the LP method over value iteration is that it avoids the need for iterative learning with the geometric convergence exhibited by value iteration. Given the dramatic strides in the speed of linear programming solvers over the last decade, the relative performance of value iteration over the linear programming method is an unresolved question. However, this question only arises for problems with relatively small state and action spaces. While a linear program with 50,000 constraints is considered large, dynamic programs with 50,000 states and actions tend to be relatively small problems.

14.11 LINEAR QUADRATIC REGULATION

Easily the best known problem in optimal control is the problem known as *linear quadratic regulation*. This is a problem known only to the controls community, and for this reason, we are going to revert to classical controls notation, which is the only way it will ever appear in any popular presentation (except that we are still going to use time t in contrast with the more typical "k" used by the controls community). As you read this model, it is best to think of it in the context of a problem such as managing a robot or rocket where

x_t = the state vector, giving the location (in two or three dimensions), and velocity (again, in two or three dimensions),

u_t = the control vector, giving the force applied to each of the two (or three) dimensions.

The state evolves according to linear equations

$$x_{t+1} = A_t x_t + B_t u_t, \tag{14.43}$$

which captures the effect of force on location and velocity. Our goal is to find controls $u_1, \ldots, u_T$ to minimize the costs given by

$$\begin{aligned} C_t(x_t, u_t) &= \frac{1}{2} x_t^T Q_t x_t + u_t^T R_T u_t, \;\; t = 1, \ldots, T-1, \\ C_T(x_T) &= \frac{1}{2} x_T^T S_T x_T, \end{aligned}$$

where Q_t, R_t and S_T are symmetric, positive semidefinite matrices. Next form the objective function

$$J = \frac{1}{2} \sum_{t=1}^{T} C_t(x_t, u_t) + \frac{1}{2} C_T(x_T),$$

subject to the system dynamics given by (14.43). Note that x_t and u_t are unconstrained. We are going to use the principle of *Lagrangian relaxation* and relax (14.43) and add the deviation to the objective function, giving us the *Lagrangian* (this is a standard optimization technique)

$$L(u_1,\ldots,u_T,\lambda) = \frac{1}{2}\sum_{t=1}^{T} C_t(x_t,u_t) + \frac{1}{2}C_T(x_T) + \lambda_{t+1}(A_t x_t + B_t u_t - x_{t+1}). \quad (14.44)$$

The controls community then defines a portion of the Lagrangian, called the Hamiltonian, which is

$$H_t = \frac{1}{2}x_t^T Q_t x_t + u_t^T R_T u_t + \lambda_{t+1}(A_t x_t + B_t u_t).$$

Differentiating (14.44) with respect to λ_{t+1} and setting the derivative equal to zero (which would be true at optimality) returns our transition equation by setting

$$\frac{\partial L(u_1,\ldots,u_T,\lambda)}{\partial \lambda_{t+1}} = (A_t x_t + B_t u_t - x_{t+1}) = 0,$$

from which we regain (14.43) (known as the *state equations*). Students of linear programming will recognize that λ_t as a dual variable.

Then, we obtain the *costate equations* by differentiating H_t with respect to x_t giving us

$$\lambda_t = \frac{\partial H_t}{\partial x_t} = Q_t x_t + A_t^T \lambda_{t+1}, \quad (14.45)$$

which essentially gives us our dual variables.

This system is solved essentially using the derivatives given in section 12.7. From these, we can derive the following feedback equations that are solved backward in time starting from a given S_T (typically determined by where we want our device to end up):

$$S_t = A_t^T \left[S_{t+1} - S_{t+1}B_t(B_t^T S_{t+1}B_t + R_t)^{-1}B_t^T S_{t+1}\right] A_t + Q_t. \quad (14.46)$$

We can then compute

$$K_t = \left(B_t^T S_{t+1}B_t + R_t\right)^{-1} B_t^T S_{t+1}A_t. \quad (14.47)$$

Our optimal control is then given by

$$u_t^* = -K_t x_t. \quad (14.48)$$

We note that these derivations have all been done in the context of a deterministic problem. One way to introduce uncertainty is with additive noise

$$x_{t+1} = A_t x_t + B_t u_t + w_t, \quad (14.49)$$

where w_t is random at time t (this is the classical style of the optimal control community - we use W_{t+1} elsewhere in this book). Additive noise can enter, for example, when exogenous forces (such as wind) interfere with the evolution of the system over time.

Adding additive noise as we do in equation (14.49) does not change our solution. When it is added to the Hamiltonian, we take expectations and since we will assume that $\mathbb{E}w_t = 0$, the noise term just drops out.

This is again a rare case of a truly optimal policy, which is very dependent on the characteristics of this problem:

- The quadratic form of the cost function (quadratic in both the state x_t and control u_t).
- The fact that it is completely unconstrained.

What is especially important about the optimal control given by (14.48) is that it is *linear* in the controls u_t. This suggests a starting point for problems that do not satisfy all of these conditions. One strategy that has been successfully applied is to assume that the policy is locally linear, which is that it is linear in the controls, but with coefficients that are defined only over specific regions.

14.12 WHY DOES IT WORK?**

The theory of Markov decision processes is especially elegant for students who enjoy probabilistic mathematics. While not needed for computational work, an understanding of why they work will provide a deeper appreciation of the properties of these problems.

Section 14.12.1 provides a proof that the optimal value function satisfies the optimality equations. Section 14.12.2 proves convergence of the value iteration algorithm. Section 14.12.3 then proves conditions under which value iteration increases or decreases monotonically to the optimal solution. Then, section 14.12.4 proves the bound on the error when value iteration satisfies the termination criterion given in section 14.6.3. Section 14.12.5 closes with a discussion of deterministic and randomized policies, along with a proof that deterministic policies are always at least as good as a randomized policy.

14.12.1 The optimality equations

Until now, we have been presenting the optimality equations as though they were a fundamental law of some sort. To be sure, they can easily look as though they were intuitively obvious, but it is still important to establish the relationship between the original optimization problem and the optimality equations. Since these equations are the foundation of dynamic programming, it seems beholden on us to work through the steps of proving that they are actually true.

We start by remembering the original optimization problem:

$$F_t^\pi(S_t) = \mathbb{E}\left\{\sum_{t'=t}^{T-1} C_{t'}(S_{t'}, A_{t'}^\pi(S_{t'})) + C_T(S_T)|S_t\right\}. \tag{14.50}$$

Since (14.50) is, in general, exceptionally difficult to solve, we resort to the optimality equations

$$V_t^\pi(S_t) = C_t(S_t, A_t^\pi(S_t)) + \mathbb{E}\left\{V_{t+1}^\pi(S_{t+1})|S_t\right\}. \tag{14.51}$$

Our challenge is to show that these are the same. In order to establish this result, it is going to help if we first prove the following:

Lemma 14.12.1. *Let S_t be a state variable that captures the relevant history up to time t, and let $F_{t'}(S_{t+1})$ be some function measured at time $t' \geq t+1$ conditioned on the random*

variable S_{t+1}. Then

$$\mathbb{E}\left[\mathbb{E}\{F_{t'}|S_{t+1}\}|S_t\right] = \mathbb{E}\left[F_{t'}|S_t\right]. \quad (14.52)$$

Proof: This lemma is variously known as the law of iterated expectations or the tower property. Assume, for simplicity, that $F_{t'}$ is a discrete, finite random variable that takes outcomes in $\mathcal{F}$. We start by writing

$$\mathbb{E}\{F_{t'}|S_{t+1}\} = \sum_{f\in\mathcal{F}} f\mathbb{P}(F_{t'} = f|S_{t+1}). \quad (14.53)$$

Recognizing that S_{t+1} is a random variable, we may take the expectation of both sides of (14.53), conditioned on S_t as follows:

$$\mathbb{E}\left[\mathbb{E}\{F_{t'}|S_{t+1}\}|S_t\right] = \sum_{S_{t+1}\in\mathcal{S}}\sum_{f\in\mathcal{F}} f\mathbb{P}(F_{t'} = f|S_{t+1}, S_t)\mathbb{P}(S_{t+1} = S_{t+1}|S_t). \quad (14.54)$$

First, we observe that we may write $\mathbb{P}(F_{t'} = f|S_{t+1}, S_t) = \mathbb{P}(F_{t'} = f|S_{t+1})$, because conditioning on S_{t+1} makes all prior history irrelevant. Next, we can reverse the summations on the right-hand side of (14.54) (some technical conditions have to be satisfied to do this, but these are satisfied if the random variables are discrete and finite). This means

$$\begin{aligned}
\mathbb{E}\left[\mathbb{E}\{F_{t'}|S_{t+1} = S_{t+1}\}|S_t\right] &= \sum_{f\in\mathcal{F}}\sum_{S_{t+1}\in\mathcal{S}} f\mathbb{P}(F_{t'} = f|S_{t+1}, S_t)\mathbb{P}(S_{t+1} = S_{t+1}|S_t) \\
&= \sum_{f\in\mathcal{F}} f \sum_{S_{t+1}\in\mathcal{S}} \mathbb{P}(F_{t'} = f, S_{t+1}|S_t) \\
&= \sum_{f\in\mathcal{F}} f\mathbb{P}(F_{t'} = f|S_t) \\
&= \mathbb{E}\left[F_{t'}|S_t\right],
\end{aligned}$$

which proves our result. Note that the essential step in the proof occurs in the first step when we add S_t to the conditioning. □

We are now ready to show the following:

Proposition 14.12.1. $F_t^\pi(S_t) = V_t^\pi(S_t)$.

Proof: To prove that (14.50) and (14.51) are equal, we use a standard trick in dynamic programming: proof by induction. Clearly, $F_T^\pi(S_T) = V_T^\pi(S_T) = C_T(S_T)$. Next, assume that it holds for $t+1, t+2, \ldots, T$. We want to show that it is true for t. This means that we can write

$$V_t^\pi(S_t) = C_t(S_t, A_t^\pi(S_t)) + \mathbb{E}\left[\underbrace{\mathbb{E}\left\{\sum_{t'=t+1}^{T-1} C_{t'}(S_{t'}, A_{t'}^\pi(S_{t'})) + C_t(S_T(\omega))\,\middle|\, S_{t+1}\right\}}_{F_{t+1}^\pi(S_{t+1})}\,\middle|\, S_t\right].$$

We then use lemma 14.12.1 to write $\mathbb{E}\left[\mathbb{E}\left\{\ldots|S_{t+1}\right\}|S_t\right] = \mathbb{E}\left[\ldots|S_t\right]$. Hence,

$$V_t^\pi(S_t) = C_t(S_t, A_t^\pi(S_t)) + \mathbb{E}\left[\sum_{t'=t+1}^{T-1} C_{t'}(S_{t'}, A_{t'}^\pi(S_{t'})) + C_t(S_T)|S_t\right].$$

When we condition on S_t, $A_t^\pi(S_t)$ (and therefore $C_t(S_t, A_t^\pi(S_t))$) is deterministic, so we can pull the expectation out to the front giving

$$\begin{aligned} V_t^\pi(S_t) &= \mathbb{E}\left[\sum_{t'=t}^{T-1} C_{t'}(S_{t'}, y_{t'}(S_{t'})) + C_t(S_T)|S_t\right] \\ &= F_t^\pi(S_t), \end{aligned}$$

which proves our result. □

Using equation (14.51), we have a backward recursion for calculating $V_t^\pi(S_t)$ for a given policy π. Now that we can find the expected reward for a given π, we would like to find the best π. That is, we want to find

$$F_t^*(S_t) = \max_{\pi\in\Pi} F_t^\pi(S_t).$$

If the set Π is infinite, we replace the "$\max$" with "$\sup$." We solve this problem by solving the optimality equations. These are

$$V_t(S_t) = \max_{a\in\mathcal{A}} \big(C_t(S_t, a) + \sum_{s'\in\mathcal{S}} p_t(s'|S_t, a)V_{t+1}(s')\big). \tag{14.55}$$

We are claiming that if we find the set of $V's$ that solves (14.55), then we have found the policy that optimizes F_t^π. We state this claim formally as:

Theorem 14.12.1. *Let $V_t(S_t)$ be a solution to equation (14.55). Then*

$$\begin{aligned} F_t^* &= V_t(S_t) \\ &= \max_{\pi\in\Pi} F_t^\pi(S_t). \end{aligned}$$

Proof: The proof is in two parts. First, we show by induction that $V_t(S_t) \geq F_t^*(S_t)$ for all $S_t \in \mathcal{S}$ and $t = 0, 1, \ldots, T-1$. Then, we show that the reverse inequality is true, which gives us the result.

Part 1:

We resort again to our proof by induction. Since $V_T(S_T) = C_t(S_T) = F_T^\pi(S_T)$ for all S_T and all $\pi \in \Pi$, we get that $V_T(S_T) = F_T^*(S_T)$.

Assume that $V_{t'}(S_{t'}) \geq F_{t'}^*(S_{t'})$ for $t' = t+1, t+2, \ldots, T$, and let π be an arbitrary policy. For $t' = t$, the optimality equation tells us

$$V_t(S_t) = \max_{a\in\mathcal{A}} \left(C_t(S_t, a) + \sum_{s'\in\mathcal{S}} p_t(s'|S_t, a)V_{t+1}(s')\right).$$

By the induction hypothesis, $F_{t+1}^*(s) \leq V_{t+1}(s)$, so we get

$$V_t(S_t) \geq \max_{a\in\mathcal{A}} \left(C_t(S_t, a) + \sum_{s'\in\mathcal{S}} p_t(s'|S_t, a)F_{t+1}^*(s')\right).$$

Of course, we have that $F_{t+1}^*(s) \geq F_{t+1}^\pi(s)$ for an arbitrary π. Also let $A^\pi(S_t)$ be the decision that would be chosen by policy π when in state S_t. Then

$$\begin{aligned} V_t(S_t) &\geq \max_{a\in\mathcal{A}} \left(C_t(S_t, a) + \sum_{s'\in\mathcal{S}} p_t(s'|S_t, a)F_{t+1}^\pi(s')\right) \\ &\geq C_t(S_t, A^\pi(S_t)) + \sum_{s'\in\mathcal{S}} p_t(s'|S_t, A^\pi(S_t))F_{t+1}^\pi(s') \\ &= F_t^\pi(S_t). \end{aligned}$$

This means

$$V_t(S_t) \geq F_t^{\pi}(S_t) \quad \text{for all } \pi \in \Pi,$$

which proves part 1.

Part 2:

Now we are going to prove the inequality from the other side. Specifically, we want to show that for any $\epsilon > 0$ there exists a policy π that satisfies

$$F_t^{\pi}(S_t) + (T-t)\epsilon \geq V_t(S_t). \tag{14.56}$$

To do this, we start with the definition

$$V_t(S_t) = \max_{a \in \mathcal{A}} \left(C_t(S_t, a) + \sum_{s' \in \mathcal{S}} p_t(s'|S_t, a) V_{t+1}(s') \right). \tag{14.57}$$

We may let $a_t(S_t)$ be the decision rule that solves (14.57). This rule corresponds to the policy π. In general, the set $\mathcal{A}$ may be infinite, whereupon we have to replace the "$\max$" with a "$\sup$" and handle the case where an optimal decision may not exist. For this case, we know that we can design a decision rule $a_t(S_t)$ that returns a decision a that satisfies

$$V_t(S_t) \leq C_t(S_t, a) + \sum_{s' \in \mathcal{S}} p_t(s'|S_t, a) V_{t+1}(s') + \epsilon. \tag{14.58}$$

We can prove (14.56) by induction. We first note that (14.56) is true for $t = T$ since $F_T^{\pi}(S_t) = V_T(S_T)$. Now assume that it is true for $t' = t+1, t+2, \ldots, T$. We already know that

$$F_t^{\pi}(S_t) = C_t(S_t, A^{\pi}(S_t)) + \sum_{s' \in \mathcal{S}} p_t(s'|S_t, A^{\pi}(S_t)) F_{t+1}^{\pi}(s').$$

We can use our induction hypothesis which says $F_{t+1}^{\pi}(s') \geq V_{t+1}(s') - (T - (t+1))\epsilon$ to get

$$\begin{aligned}
F_t^{\pi}(S_t) &\geq C_t(S_t, A^{\pi}(S_t)) + \sum_{s' \in \mathcal{S}} p_t(s'|S_t, A^{\pi}(S_t))[V_{t+1}(s') - (T-(t+1))\epsilon] \\
&= C_t(S_t, A^{\pi}(S_t)) + \sum_{s' \in \mathcal{S}} p_t(s'|S_t, A^{\pi}(S_t)) V_{t+1}(s') \\
&\quad - \sum_{s' \in \mathcal{S}} p_t(s'|S_t, A^{\pi}(S_t)) \left[(T-t-1)\epsilon\right] \\
&= \left\{ C_t(S_t, A^{\pi}(S_t)) + \sum_{s' \in \mathcal{S}} p_t(s'|S_t, A^{\pi}(S_t)) V_{t+1}(s') + \epsilon \right\} - (T-t)\epsilon.
\end{aligned}$$

Now, using equation (14.58), we replace the term in brackets with the smaller $V_t(S_t)$ (equation (14.58)):

$$F_t^{\pi}(S_t) \geq V_t(S_t) - (T-t)\epsilon,$$

which proves the induction hypothesis. We have shown that

$$F_t^*(S_t) + (T-t)\epsilon \geq F_t^{\pi}(S_t) + (T-t)\epsilon \geq V_t(S_t) \geq F_t^*(S_t).$$

This proves the result. □

Now we know that solving the optimality equations also gives us the optimal value function. This is our most powerful result because we can solve the optimality equations for many problems that cannot be solved any other way.

14.12.2 Convergence of value iteration

We now undertake the proof that the basic value function iteration converges to the optimal solution. This is not only an important result, it is also an elegant one that brings some powerful theorems into play. The proof is also quite short. However, we will need some mathematical preliminaries:

Definition 14.12.1. *Let $\mathcal{V}$ be a set of (bounded, real-valued) functions and define the norm of v by:*

$$\|v\| \quad = \quad \sup_{s \in \mathcal{S}} v(s)$$

where we replace the "sup" with a "max" when the state space is finite. Since $\mathcal{V}$ is closed under addition and scalar multiplication and has a norm, it is a **normed linear space**.

Definition 14.12.2. *$T : \mathcal{V} \rightarrow \mathcal{V}$ is a* **contraction mapping** *if there exists a γ, $0 \leq \gamma < 1$ such that:*

$$\|Tv - Tu\| \quad \leq \quad \gamma \|v - u\|.$$

Definition 14.12.3. *A sequence $v^n \in \mathcal{V},\ n = 1, 2, \ldots$ is said to be a* **Cauchy sequence** *if for all $\epsilon > 0$, there exists N such that for all $n, m \geq N$:*

$$\|v^n - v^m\| < \epsilon.$$

Definition 14.12.4. *A normed linear space is* **complete** *if every Cauchy sequence contains a limit point in that space.*

Definition 14.12.5. *A* **Banach space** *is a complete normed linear space.*

Definition 14.12.6. *We define the norm of a matrix Q as*

$$\|Q\| \quad = \quad \max_{s \in \mathcal{S}} \sum_{j \in \mathcal{S}} |q(j|s)|,$$

that is, the largest row sum of the matrix. If Q is a one-step transition matrix, then $\|Q\| = 1$.

Definition 14.12.7. *The* **triangle inequality** *means that given two vectors $a, b \in \Re^n$:*

$$\|a + b\| \quad \leq \quad \|a\| + \|b\|.$$

The triangle inequality is commonly used in proofs because it helps us establish bounds between two solutions (and in particular, between a solution and the optimum).

We now state and prove one of the famous theorems in applied mathematics and then use it immediately to prove convergence of the value iteration algorithm.

Theorem 14.12.2. *(Banach Fixed-Point Theorem) Let $\mathcal{V}$ be a Banach space, and let $T : \mathcal{V} \rightarrow \mathcal{V}$ be a contraction mapping. Then:*

(a) There exists a unique $v^* \in \mathcal{V}$ such that $Tv^* = v^*$.

(b) For an arbitrary $v^0 \in \mathcal{V}$, the sequence v^n defined by: $v^{n+1} = Tv^n = T^{n+1}v^0$ converges to v^*.

Proof: We start by showing that the distance between two vectors v^n and v^{n+m} goes to zero for sufficiently large n and by writing the difference $v^{n+m} - v^n$ using

$$\begin{aligned} v^{n+m} - v^n &= v^{n+m} - v^{n+m-1} + v^{n+m-1} - \cdots - v^{n+1} + v^{n+1} - v^n \\ &= \sum_{k=0}^{m-1} (v^{n+k+1} - v^{n+k}). \end{aligned}$$

Taking norms of both sides and invoking the triangle inequality gives

$$\begin{aligned} \|v^{n+m} - v^n\| &= \| \sum_{k=0}^{m-1} (v^{n+k+1} - v^{n+k})\| \\ &\leq \sum_{k=0}^{m-1} \|(v^{n+k+1} - v^{n+k})\| \\ &= \sum_{k=0}^{m-1} \|(T^{n+k}v^1 - T^{n+k}v^0)\| \\ &\leq \sum_{k=0}^{m-1} \gamma^{n+k}\|v^1 - v^0\| \\ &= \frac{\gamma^n(1-\gamma^m)}{(1-\gamma)}\|v^1 - v^0\|. \end{aligned} \tag{14.59}$$

Since $\gamma < 1$, for sufficiently large n the right-hand side of (14.59) can be made arbitrarily small, which means that v^n is a Cauchy sequence. Since $\mathcal{V}$ is *complete*, it must be that v^n has a limit point v^*. From this we conclude

$$\lim_{n\to\infty} v^n \to v^*. \tag{14.60}$$

We now want to show that v^* is a fixed point of the mapping T. To show this, we observe

$$\begin{aligned} 0 &\leq \|Tv^* - v^*\| && (14.61)\\ &= \|Tv^* - v^n + v^n - v^*\| && (14.62)\\ &\leq \|Tv^* - v^n\| + \|v^n - v^*\| && (14.63)\\ &= \|Tv^* - Tv^{n-1}\| + \|v^n - v^*\| && (14.64)\\ &\leq \gamma\|v^* - v^{n-1}\| + \|v^n - v^*\|. && (14.65) \end{aligned}$$

Equation (14.61) comes from the properties of a norm. We play our standard trick in (14.62) of adding and subtracting a quantity (in this case, v^n), which sets up the triangle inequality in (14.63). Using $v^n = Tv^{n-1}$ gives us (14.64). The inequality in (14.65) is based on the assumption of the theorem that T is a contraction mapping. From (14.60), we know that

$$\lim_{n\to\infty} \|v^* - v^{n-1}\| = \lim_{n\to\infty} \|v^n - v^*\| = 0. \tag{14.66}$$

Combining (14.61), (14.65), and (14.66) gives

$$0 \leq \|Tv^* - v^*\| \leq 0,$$

from which we conclude

$$\|Tv^* - v^*\| \quad = \quad 0,$$

which means that $Tv^* = v^*$.

We can prove uniqueness by contradiction. Assume that there are two limit points that we represent as v^* and u^*. The assumption that T is a contraction mapping requires that

$$\|Tv^* - Tu^*\| \leq \gamma \|v^* - u^*\|.$$

But, if v^* and u^* are limit points, then $Tv^* = v^*$ and $Tu^* = u^*$, which means

$$\|v^* - u^*\| \leq \gamma \|v^* - u^*\|.$$

Since $\gamma < 1$, this is a contradiction, which means that it must be true that $v^* = u^*$. □

We can now show that the value iteration algorithm converges to the optimal solution if we can establish that $\mathcal{M}$ is a contraction mapping. So we need to show the following:

Proposition 14.12.2. *If $0 \leq \gamma < 1$, then $\mathcal{M}$ is a contraction mapping on $\mathcal{V}$.*

Proof: Let $u, v \in \mathcal{V}$ and assume that $\mathcal{M}v \geq \mathcal{M}u$ where the inequality is applied elementwise. For a particular state s let

$$a_s^*(v) \in \arg\max_{a \in \mathcal{A}} \left(C(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s,a) v(s') \right)$$

where we assume that a solution exists. Then

$$\begin{aligned}
0 &\leq \mathcal{M}v(s) - \mathcal{M}u(s) && (14.67)\\
&= C(s, a_s^*(v)) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a_s^*(v)) v(s') \\
&\quad - \left(C(s, a_s^*(u)) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a_s^*(u)) u(s') \right) && (14.68)\\
&\leq C(s, a_s^*(v)) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a_s^*(v)) v(s') \\
&\quad - \left(C(s, a_s^*(v)) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a_s^*(v)) u(s') \right) && (14.69)\\
&= \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a_s^*(v)) [v(s') - u(s')] && (14.70)\\
&\leq \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a_s^*(v)) \|v - u\| && (14.71)\\
&= \gamma \|v - u\| \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a_s^*(v)) && (14.72)\\
&= \gamma \|v - u\|. && (14.73)
\end{aligned}$$

Equation (14.67) is true by assumption, while (14.68) holds by definition. The inequality in (14.69) holds because $a_s^*(v)$ is not optimal when the value function is u, giving a reduced value in the second set of parentheses. Equation (14.70) is a simple reduction of (14.69).

Equation (14.71) forms an upper bound because the definition of $\|v-u\|$ is to replace all the elements $[v(s)-u(s)]$ with the largest element of this vector. Since this is now a vector of constants, we can pull it outside of the summation, giving us (14.72), which then easily reduces to (14.73) because the probabilities add up to one.

This result states that if $\mathcal{M}v(s) \geq \mathcal{M}u(s)$, then $\mathcal{M}v(s) - \mathcal{M}u(s) \leq \gamma|v(s)-u(s)|$. If we start by assuming that $\mathcal{M}v(s) \leq \mathcal{M}u(s)$, then the same reasoning produces $\mathcal{M}v(s) - \mathcal{M}u(s) \geq -\gamma|v(s)-u(s)|$. This means that we have

$$|\mathcal{M}v(s) - \mathcal{M}u(s)| \leq \gamma|v(s)-u(s)| \tag{14.74}$$

for *all* states $s \in \mathcal{S}$. From the definition of our norm, we can write

$$\begin{aligned}\sup_{s\in\mathcal{S}} |\mathcal{M}v(s) - \mathcal{M}u(s)| &= \|\mathcal{M}v - \mathcal{M}u\| \\ &\leq \gamma\|v-u\|.\end{aligned}$$

This means that $\mathcal{M}$ is a contraction mapping, which means that the sequence v^n generated by $v^{n+1} = \mathcal{M}v^n$ converges to a unique limit point v^* that satisfies the optimality equations. □

14.12.3 Monotonicity of value iteration

Infinite horizon dynamic programming provides a compact way to study the theoretical properties of these algorithms. The insights gained here are applicable to problems even when we cannot apply this model, or these algorithms, directly.

We assume throughout our discussion of infinite horizon problems that the reward function is bounded over the domain of the state space. This assumption is virtually always satisfied in practice, but notable exceptions exist. For example, the assumption is violated if we are maximizing a utility function that depends on the log of the resources we have at hand (the resources may be bounded, but the function is unbounded if the resources are allowed to hit zero).

Our first result establishes a monotonicity property that can be exploited in the design of an algorithm.

Theorem 14.12.3. *For a vector $v \in \mathcal{V}$:*

(a) If v satisfies $v \geq \mathcal{M}v$, then $v \geq v^*$.

(b) If v satisfies $v \leq \mathcal{M}v$, then $v \leq v^*$.

(c) If v satisfies $v = \mathcal{M}v$, then v is the unique solution to this system of equations and $v = v^*$.

Proof: Part (a) requires that

$$\begin{aligned} v &\geq \max_{\pi\in\Pi}\{c^\pi + \gamma P^\pi v\} && (14.75)\\ &\geq c^{\pi_0} + \gamma P^{\pi_0} v && (14.76)\\ &\geq c^{\pi_0} + \gamma P^{\pi_0}\left(c^{\pi_1} + \gamma P^{\pi_1} v\right) && (14.77)\\ &= c^{\pi_0} + \gamma P^{\pi_0} c^{\pi_1} + \gamma^2 P^{\pi_0} P^{\pi_1} v. \end{aligned}$$

Equation (14.75) is true by assumption (part (a) of the theorem) and equation (14.76) is true because π_0 is some policy that is not necessarily optimal for the vector v. Using

similar reasoning, equation (14.77) is true because π_1 is another policy which, again, is not necessarily optimal. Using $P^{\pi,(t)} = P^{\pi_0}P^{\pi_1}\cdots P^{\pi_t}$, we obtain by induction

$$v \geq c^{\pi_0} + \gamma P^{\pi_0}c^{\pi_1} + \cdots + \gamma^{t-1}P^{\pi_0}P^{\pi_1}\cdots P^{\pi_{t-1}}c^{\pi_t} + \gamma^t P^{\pi,(t)}v. \quad (14.78)$$

Recall that

$$v^\pi = \sum_{t=0}^{\infty}\gamma^t P^{\pi,(t)}c^{\pi_t}. \quad (14.79)$$

Breaking the sum in (14.79) into two parts allows us to rewrite the expansion in (14.78) as

$$v \geq v^\pi - \sum_{t'=t+1}^{\infty}\gamma^{t'}P^{\pi,(t')}c^{\pi_{t'+1}} + \gamma^t P^{\pi,(t)}v. \quad (14.80)$$

Taking the limit of both sides of (14.80) as $t \to \infty$ gives us

$$v \geq \lim_{t\to\infty} v^\pi - \sum_{t'=t+1}^{\infty}\gamma^{t'}P^{\pi,(t')}c^{\pi_{t'+1}} + \gamma^t P^{\pi,(t)}v \quad (14.81)$$

$$\geq v^\pi \quad \forall\pi \in \Pi. \quad (14.82)$$

The limit in (14.81) exists as long as the reward function c^π is bounded and $\gamma < 1$. Because (14.82) is true for all $\pi \in \Pi$, it is also true for the optimal policy, which means that

$$\begin{aligned} v &\geq v^{\pi*} \\ &= v^*, \end{aligned}$$

which proves part (a) of the theorem. Part (b) can be proved in an analogous way. Parts (a) and (b) mean that $v \geq v^*$ and $v \leq v^*$. If $v = \mathcal{M}v$, then we satisfy the preconditions of both parts (a) and (b), which means they are both true and therefore we must have $v = v^*$. □

This result means that if we start with a vector that is higher than the optimal vector, then we will decline monotonically to the optimal solution (almost – we have not quite proven that we actually get to the optimal). Alternatively, if we start below the optimal vector, we will rise to it. Note that it is not always easy to find a vector v that satisfies either condition (a) or (b) of the theorem. In problems where the rewards can be positive and negative, this can be tricky.

14.12.4 Bounding the error from value iteration

We now wish to establish a bound on our error from value iteration, which will establish our stopping rule. We propose two bounds: one on the value function estimate that we terminate with and one for the long-run value of the decision rule that we terminate with. To define the latter, let π^ϵ be the policy that satisfies our stopping rule, and let v^{π^ϵ} be the infinite horizon value of following policy π^ϵ.

Theorem 14.12.4. *If we apply the value iteration algorithm with stopping parameter ϵ and the algorithm terminates at iteration n with value function v^{n+1}, then*

$$\|v^{n+1} - v^*\| \leq \epsilon/2, \quad (14.83)$$

and

$$\|v^{\pi^\epsilon} - v^*\| \leq \epsilon. \tag{14.84}$$

Proof: We start by writing

$$\begin{aligned} \|v^{\pi^\epsilon} - v^*\| &= \|v^{\pi^\epsilon} - v^{n+1} + v^{n+1} - v^*\| \\ &\leq \|v^{\pi^\epsilon} - v^{n+1}\| + \|v^{n+1} - v^*\|. \end{aligned} \tag{14.85}$$

Recall that π^ϵ is the policy that solves $\mathcal{M}v^{n+1}$, which means that $\mathcal{M}^{\pi^\epsilon}v^{n+1} = \mathcal{M}v^{n+1}$. This allows us to rewrite the first term on the right-hand side of (14.85) as

$$\begin{aligned} \|v^{\pi^\epsilon} - v^{n+1}\| &= \|\mathcal{M}^{\pi^\epsilon}v^{\pi^\epsilon} - \mathcal{M}v^{n+1} + \mathcal{M}v^{n+1} - v^{n+1}\| \\ &\leq \|\mathcal{M}^{\pi^\epsilon}v^{\pi^\epsilon} - \mathcal{M}v^{n+1}\| + \|\mathcal{M}v^{n+1} - v^{n+1}\| \\ &= \|\mathcal{M}^{\pi^\epsilon}v^{\pi^\epsilon} - \mathcal{M}^{\pi^\epsilon}v^{n+1}\| + \|\mathcal{M}v^{n+1} - \mathcal{M}v^n\| \\ &\leq \gamma\|v^{\pi^\epsilon} - v^{n+1}\| + \gamma\|v^{n+1} - v^n\|. \end{aligned}$$

Solving for $\|v^{\pi^\epsilon} - v^{n+1}\|$ gives

$$\|v^{\pi^\epsilon} - v^{n+1}\| \leq \frac{\gamma}{1-\gamma}\|v^{n+1} - v^n\|.$$

We can use similar reasoning applied to the second term in equation (14.85) to show that

$$\|v^{n+1} - v^*\| \leq \frac{\gamma}{1-\gamma}\|v^{n+1} - v^n\|. \tag{14.86}$$

The value iteration algorithm stops when $\|v^{n+1} - v^n\| \leq \epsilon(1-\gamma)/2\gamma$. Substituting this in (14.86) gives

$$\|v^{n+1} - v^*\| \leq \frac{\epsilon}{2}. \tag{14.87}$$

Recognizing that the same bound applies to $\|v^{\pi^\epsilon} - v^{n+1}\|$ and combining these with (14.85) gives us

$$\|v^{\pi^\epsilon} - v^*\| \leq \epsilon,$$

which completes our proof. □

14.12.5 Randomized policies

We have implicitly assumed that for each state, we want a single action. An alternative would be to choose a policy probabilistically from a family of policies. If a state produces a single action, we say that we are using a *deterministic policy*. If we are randomly choosing an action from a set of actions probabilistically, we say we are using a *randomized policy*.

Randomized policies may arise because of the nature of the problem. For example, you wish to purchase something at an auction, but you are unable to attend yourself. You may have a simple rule ("purchase it as long as the price is under a specific amount") but you cannot assume that your representative will apply the same rule. You can choose a representative, and in doing so you are effectively choosing the probability distribution from which the action will be chosen.

Behaving randomly also plays a role in two-player games. If you make the same decision each time in a particular state, your opponent may be able to predict your behavior and gain an advantage. For example, as an institutional investor you may tell a bank that you not willing to pay any more than \$14 for a new offering of stock, while in fact you are willing to pay up to \$18. If you always bias your initial prices by \$4, the bank will be able to guess what you are willing to pay.

When we can only influence the likelihood of an action, then we have an instance of a randomized MDP. Let

$q_t^\pi(a|S_t) =$ The probability that decision a will be taken at time t given state S_t and policy π (more precisely, decision rule A^π).

In this case, our optimality equations look like

$$V_t^*(S_t) = \max_{\pi \in \Pi^{MR}} \sum_{a \in \mathcal{A}} \left[q_t^\pi(a|S_t) \left(C_t(S_t,a) + \sum_{s' \in \mathcal{S}} p_t(s'|S_t,a) V_{t+1}^*(s') \right) \right]. \quad (14.88)$$

Now let us consider the single best action that we could take. Calling this a^*, we can find it using

$$a^* = \arg\max_{a \in \mathcal{A}} \left[C_t(S_t,a) + \sum_{s' \in \mathcal{S}} p_t(s'|S_t,a) V_{t+1}^*(s') \right].$$

This means that

$$C_t(S_t,a^*) + \sum_{s' \in \mathcal{S}} p_t(s'|S_t,a^*) V_{t+1}^*(s') \quad \geq \quad C_t(S_t,a) + \sum_{s' \in \mathcal{S}} p_t(s'|S_t,a) V_{t+1}^*(s') \quad (14.89)$$

for all $a \in \mathcal{A}$. Substituting (14.89) back into (14.88) gives us

$$\begin{aligned} V_t^*(S_t) &= \max_{\pi \in \Pi^{MR}} \sum_{a \in \mathcal{A}} \left[q_t^\pi(a|S_t) \left(C_t(S_t,a) + \sum_{s' \in \mathcal{S}} p_t(s'|S_t,a) V_{t+1}^*(s') \right) \right] \\ &\leq \max_{\pi \in \Pi^{MR}} \sum_{a \in \mathcal{A}} \left[q_t^\pi(a|S_t) \left(C_t(S_t,a^*) + \sum_{s' \in \mathcal{S}} p_t(s'|S_t,a^*) V_{t+1}^*(s') \right) \right] \\ &= C_t(S_t,a^*) + \sum_{s' \in \mathcal{S}} p_t(s'|S_t,a^*) V_{t+1}^*(s'). \end{aligned}$$

What this means is that if you have a choice between picking exactly the action you want versus picking a probability distribution over potentially optimal and nonoptimal actions, you would always prefer to pick exactly the best action. Clearly, this is not a surprising result.

The value of randomized policies arise primarily in two-person games, where one player tries to anticipate the actions of the other player. In such situations, part of the state variable is the estimate of what the other play will do when the game is in a particular state. By randomizing his behavior, a player reduces the ability of the other player to anticipate his moves.

14.13 BIBLIOGRAPHIC NOTES

This chapter presents the classic view of Markov decision processes, for which the literature is extensive. Beginning with the seminal text of Bellman (Bellman (1957)), there have been numerous, significant textbooks on the subject, including Howard (1960), Nemhauser (1966), White (1969), Derman (1970), Bellman (1971), Dreyfus & Law (1977), Dynkin & Yushkevich (1979), Denardo (1982), Ross (1983) and Heyman & Sobel (1984). As of this writing, the current high-water mark for textbooks in this area is the landmark volume by Puterman (2005). Most of this chapter is based on Puterman (2005), modified to our notational style.

Section 14.10 - The linear programming method was first proposed in Manne (1960) (see subsequent discussions in Derman (1962) and Puterman (2005)). The so-called "linear programming method" was ignored for many years because of the large size of the linear programs that were produced, but the method has seen a resurgence of interest using approximation techniques. Recent research into algorithms for solving problems using this method are discussed in section 17.10.

Section 14.11 - This section was adapted from Lewis & Vrabie (2012), section 2.2.

EXERCISES

Review questions

14.1 Discrete Markov decision processes have been studied since the 1950's as a way of solving stochastic, dynamic programs. Yet, in chapter 4, this is used as an example of a stochastic optimization problem that can be solved deterministically. Explain.

14.2 A classical inventory problem works as follows: Assume that our state variable R_t is the amount of product on hand at the end of time period t and that D_t is a random variable giving the demand during time interval $(t-1,t)$ with distribution $p_d = \mathbb{P}(D_t = d)$. The demand in time interval t must be satisfied with the product on hand at the beginning of the period. We can then order a quantity x_t at the end of period t that can be used to replenish the inventory in period $t+1$.

(a) Give the transition function that relates R_{t+1} to R_t if the order quantity is x_t (where x_t is fixed for all R_t).

(b) Give an algebraic version of the one-step transition matrix $P^\pi = \{p^\pi_{ij}\}$ where $p^\pi_{ij} = \mathbb{P}(R_{t+1} = j | R_t = i, A^\pi = x_t)$.

14.3 Repeat the previous exercise, but now assume that we have adopted a policy π that says we should order a quantity $x_t = 0$ if $R_t \geq s$ and $x_t = Q - R_t$ if $R_t < q$ (we assume that $R_t \leq Q$). Your expression for the transition matrix will now depend on our policy π (which describes both the structure of the policy and the control parameter s).

Modeling questions

14.4 Every day, a salesman visits N customers in order to sell the R identical items he has in his van. Each customer is visited exactly once and each customer buys zero or one item. Upon arrival at a customer location, the salesman quotes one of the prices $0 < p_1 \leq p_2 \leq \ldots \leq p_m$. Given that the quoted price is p_i, a customer buys an item with probability r_i. Naturally, r_i is decreasing in i. The salesman is interested in maximizing the total expected revenue for the day. Show that if $r_i p_i$ is increasing in i, then it is always optimal to quote the highest price p_m.

14.5 You need to decide when to replace your car. If you own a car of age y years, then the cost of maintaining the car that year will be $c(y)$. Purchasing a new car (in constant dollars) costs P dollars. If the car breaks down, which it will do with probability $b(y)$ (the breakdown probability), it will cost you an additional K dollars to repair it, after which you immediately sell the car and purchase a new one. At the same time, you express your enjoyment with owning a new car as a negative cost $-r(y)$ where $r(y)$ is a declining function with age. At the beginning of each year, you may choose to purchase a new car ($z = 1$) or to hold onto your old one ($z = 0$). You anticipate that you will actively drive a car for another T years.

(a) Identify all the elements of a Markov decision process for this problem.

(b) Write out the objective function which will allow you to find an optimal decision rule.

(c) Write out the one-step transition matrix.

(d) Write out the optimality equations that will allow you to solve the problem.

14.6 Describe the gambling problem in section 14.4.1 as a decision tree, assuming that we can gamble only 0, 1 or 2 dollars in each round (this is just to keep the decision tree from growing too large).

14.7 You are trying to find the best parking space to use that minimizes the time needed to get to your restaurant. There are 50 parking spaces, and you see spaces $1, 2, \ldots, 50$ in order. As you approach each parking space, you see whether it is full or empty. We assume, somewhat heroically, that the probability that each space is occupied follows an independent Bernoulli process, which is to say that each space will be occupied with probability p, but will be free with probability $1 - p$, and that each outcome is independent of the other.

It takes 2 seconds to drive past each parking space and it takes 8 seconds to walk past. That is, if we park in space n, it will require $8(50 - n)$ seconds to walk to the restaurant. Furthermore, it would have taken you $2n$ seconds to get to this space. If you get to the last space without finding an opening, then you will have to drive into a special lot down the block, adding 30 seconds to your trip.

We want to find an optimal strategy for accepting or rejecting a parking space.

(a) Give the sets of state and action spaces and the set of decision epochs.

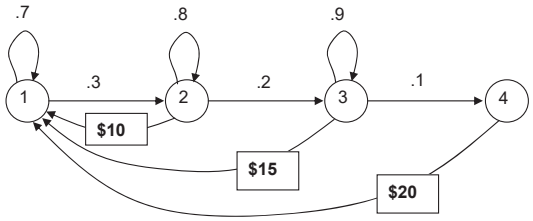
(b) Give the expected reward function for each time period and the expected final reward function.

(c) Give a formal statement of the objective function.

(d) Give the optimality equations for solving this problem.

(e) You have just looked at space 45, which was empty. There are five more spaces remaining (46 through 50). What should you do? Using $p = 0.6$, find the optimal policy by solving your optimality equations for parking spaces 46 through 50.

f) Give the optimal value of the objective function in part (e) corresponding to your optimal solution.



Computational exercises

14.8 We are going to use a very simple Markov decision process to illustrate how the initial estimate of the value function can affect convergence behavior. In fact, we are going to use a Markov reward process to illustrate the behavior because our process does not have any decisions. Assume we have a two-stage Markov chain with one-step transition matrix

$$P = \begin{bmatrix} 0.7 & 0.3 \\ 0.05 & 0.95 \end{bmatrix}.$$

The contribution from each transition from state $i \in \{1, 2\}$ to state $j \in \{1, 2\}$ is given by the matrix

$$\begin{bmatrix} 10 & 30 \\ 30 & 5 \end{bmatrix}.$$

That is, a transition from state 1 to state 2 returns a contribution of 30. Apply the value iteration algorithm for an infinite horizon problem (note that you are not choosing a decision so there is no maximization step). The calculation of the value of being in each state will depend on your previous estimate of the value of being in each state. The calculations can be easily implemented in a spreadsheet. Assume that your discount factor is .8.

(a) Plot the value of being in state 1 as a function of the number of iterations if your initial estimate of the value of being in each state is 0. Show the graph for 50 iterations of the algorithm.

(b) Repeat this calculation using initial estimates of 100.

(c) Repeat the calculation using an initial estimate of the value of being in state 1 of 100, and use 0 for the value of being in state 2. Contrast the behavior with the first two starting points.

14.9 Apply policy iteration to the problem given in exercise 14.8. Plot the average value function (that is, average the value of being in each state) after each iteration alongside the average value function found using value iteration after each iteration (for value iteration, initialize the value function to zero). Compare the computation time for one iteration of value iteration and one iteration of policy iteration.

14.10 Now apply the hybrid value-policy iteration algorithm to the problem given in exercise 14.8. Show the average value function after each major iteration (update of n) with $M = 1, 2, 3, 5, 10$. Compare the convergence rate to policy iteration and value iteration.

14.11 We have a four-state process (shown in the figure). In state 1, we will remain in the state with probability 0.7 and will make a transition to state 2 with probability 0.3. In states 2 and 3, we may choose between two policies: Remain in the state waiting for an upward transition or make the decision to return to state 1 and receive the indicated reward. In state 4, we return to state 1 immediately and receive \$20. We wish to find an optimal long run policy using a discount factor $\gamma = .8$. Set up and solve the optimality equations for this problem.

14.12 Assume that you have been applying value iteration to a four-state Markov decision process, and that you have obtained the values over iterations 8 through 12 shown in the table below (assume a discount factor of 0.90). Assume you stop after iteration 12. Give the tightest possible (valid) bounds on the optimal value of being in each state.

	Iteration				
State	8	9	10	11	12
1	7.42	8.85	9.84	10.54	11.03
2	4.56	6.32	7.55	8.41	9.01
3	11.83	13.46	14.59	15.39	15.95
4	8.13	9.73	10.85	11.63	12.18

14.13 Assume that a control limit policy exists for our shuttle problem in exercise 2 that allows us to write the optimal dispatch rule as a function of s, as in $z^\pi(s)$. We may write $r(s, z)$ as a function of one variable, the state s.

(a) Illustrate the shape of $r(s, z(s))$ by plotting it over the range $0 < s < 3M$ (since we are allowing there to be more customers than can fill one vehicle, assume that we are allowed to send $z = 0, 1, 2, \ldots$ vehicles in a single time period).

(b) Let $c = 10$, $h = 2$, and $M = 5$, and assume that $A_t = 1$ with probability 0.6 and is 0 with probability 0.4. Set up and solve a system of linear equations for the optimal value function for this problem in steady state.

Theory questions

14.14 Show that $\mathbb{P}(S_{t+\tau}|S_t)$, given that we are following a policy π (for stationary problems), is given by (14.22). [Hint: first show it for $\tau = 1, 2$ and then use inductive reasoning to show that it is true for general τ.]

14.15 Repeat the derivation in section 14.4.2 assuming that the reward for task t is $c_t\sqrt{x_t}$.

14.16 Repeat the derivation in section 14.4.2 assuming that the reward for task t is given by $\ln(x)$.

14.17 Repeat the derivation in section 14.4.2 one more time, but now assume that all you know is that the reward is continuously differentiable, monotonically increasing and concave.

14.18 What happens to the answer to the budget allocation problem in section 14.4.2 if the contribution is convex instead of concave (for example, $C_t(x_t) = x_t^2$)?

14.19 In the proof of theorem 14.12.3 we showed that if $v \geq \mathcal{M}v$, then $v \geq v^*$. Go through the steps of proving the converse, that if $v \leq \mathcal{M}v$, then $v \leq v^*$.

14.20 Theorem 14.12.3 states that if $v \leq \mathcal{M}v$, then $v \leq v^*$. Show that if $v^n \leq v^{n+1} = \mathcal{M}v^n$, then $v^{m+1} \geq v^m$ for all $m \geq n$.

14.21 Consider a finite-horizon MDP with the following properties:

- $\mathcal{S} \in \Re^n$, the action space $\mathcal{A}$ is a compact subset of $\Re^n$, $\mathcal{X}(s) = \mathcal{X}$ for all $s \in \mathcal{S}$.
- $C_t(S_t, x_t) = c_t S_t + g_t(x_t)$, where $g_t(\cdot)$ is a known scalar function, and $C_T(S_T) = c_T S_T$.
- If decision x_t is chosen when the state is S_t at time t, the next state is

$$S_{t+1} = A_t S_t + f_t(x_t) + \omega_{t+1},$$

where $f_t(\cdot)$ is scalar function, and A_t and ω_t are respectively $n \times n$ and $n \times 1$-dimensional random variables whose distributions are independent of the history of the process prior to t.

(a) Show that the optimal value function is linear in the state variable.

(b) Show that there exists an optimal policy $\pi^* = (x_1^*, \ldots, x_{T-1}^*)$ composed of constant decision functions. That is, $A_t^{\pi^*}(s) = A_t^*$ for all $s \in \mathcal{S}$ for some constant A_t^*.

14.22 Assume that you have invested R_0 dollars in the stock market which evolves according to the equation

$$R_t = \gamma R_{t-1} + \varepsilon_t$$

where ε_t is a discrete, positive random variable that is independent and identically distributed and where $0 < \gamma < 1$. If you sell the stock at the end of period t, it will earn a riskless return r until time T, which means it will evolve according to

$$R_t = (1 + r)R_{t-1}.$$

You have to sell the stock, all on the same day, some time before T.

(a) Write a dynamic programming recursion to solve the problem.

(b) Show that there exists a point in time τ such that it is optimal to sell for $t \geq \tau$, and optimal to hold for $t < \tau$.

(c) How does your answer to (b) change if you are allowed to sell only a portion of the assets in a given period? That is, if you have R_t dollars in your account, you are allowed to sell $x_t \leq R_t$ at time t.

14.23 Show that the matrix H^n in the recursive updating formula from equation (3.68)

$$\bar{\theta}^n = \bar{\theta}^{n-1} - H^n x^n \hat{\varepsilon}^n$$

reduces to $H^n = 1/n$ for the case of a single parameter (which means we are using Y =constant, with no independent variables).

14.24 A dispatcher controls a finite capacity shuttle that works as follows: In each time period, a random number A_t arrives. After the arrivals occur, the dispatcher must decide whether to call the shuttle to remove up to M customers. The cost of dispatching the shuttle is c, which is independent of the number of customers on the shuttle. Each time period that a customer waits costs h. If we let $z = 1$ if the shuttle departs and 0 otherwise, then our one-period reward function is given by

$$c_t(s, z) = cz + h[s - Mz]^+,$$

where M is the capacity of the shuttle. Show that $c_t(s, a)$ is submodular where we would like to minimize r. Note that we are representing the state of the system after the customers arrive.

14.25 Assume that a control limit policy exists for our shuttle problem in exercise 2 that allows us to write the optimal dispatch rule as a function of s, as in $z^\pi(s)$. We may write $r(s, z)$ as a function of one variable, the state s.

(a) Illustrate the shape of $r(s, z(s))$ by plotting it over the range $0 < s < 3M$ (since we are allowing there to be more customers than can fill one vehicle, assume that we are allowed to send $z = 0, 1, 2, \ldots$ vehicles in a single time period).

(b) Let $c = 10$, $h = 2$, and $M = 5$, and assume that $A_t = 1$ with probability 0.6 and is 0 with probability 0.4. Set up and solve a system of linear equations for the optimal value function for this problem in steady state.

14.26 Show that the matrix H^n in the recursive updating formula from equation (3.68)

$$\bar{\theta}^n = \bar{\theta}^{n-1} - H^n x^n \hat{\varepsilon}^n$$

reduces to $H^n = 1/n$ for the case of a single parameter (which means we are using Y =constant, with no independent variables).

Problem solving questions

14.27 You have to send a set of questionnaires to each of N population segments. The size of each population segment is given by w_i. You have a budget of B questionnaires to allocate among the population segments. If you send x_i questionnaires to segment i, you will have a sampling error proportional to

$$f(x_i) = 1/\sqrt{x_i}.$$

You want to minimize the weighted sum of sampling errors, given by

$$F(x) = \sum_{i=1}^{N} w_i f(x_i)$$

You wish to find the allocation x that minimizes $F(x)$ subject to the budget constraint $\sum_{i=1}^{N} x_i \leq B$. Set up the optimality equations to solve this problem as a dynamic program (needless to say, we are only interested in integer solutions).

14.28 An oil company will order tankers to fill a group of large storage tanks. One full tanker is required to fill an entire storage tank. Orders are placed at the beginning of each four week accounting period but do not arrive until the end of the accounting period. During this period, the company may be able to sell 0, 1 or 2 tanks of oil to one of the regional chemical companies (orders are conveniently made in units of storage tanks). The probability of a demand of 0, 1 or 2 is 0.40, 0.40 and 0.20, respectively.

A tank of oil costs \$1.6 million (M) to purchase and sells for \$2M. It costs \$0.020M to store a tank of oil during each period (oil ordered in period t, which cannot be sold until period $t + 1$, is not charged any holding cost in period t). Storage is only charged on oil that is in the tank at the beginning of the period and remains unsold during the period. It is possible to order more oil than can be stored. For example, the company may have two full storage tanks, order three more, and then only sell one. This means that at the end of the period, they will have four tanks of oil. Whenever they have more than two tanks of oil, the company must sell the oil directly from the ship for a price of \$0.70M. There is no penalty for unsatisfied demand.

An order placed in time period t must be paid for in time period t even though the order does not arrive until $t + 1$. The company uses an interest rate of 20 percent per accounting period (that is, a discount factor of 0.80).

(a) Give an expression for the one-period reward function $r(s, d)$ for being in state s and making decision d. Compute the reward function for all possible states (0, 1, 2) and all possible decisions (0, 1, 2).

(b) Find the one-step probability transition matrix when your action is to order one or two tanks of oil. The transition matrix when you order zero is given by

From-To	0	1	2
0	1	0	0
1	0.6	0.4	0
2	0.2	0.4	0.4

(c) Write out the general form of the optimality equations and solve this problem in steady state.

(d) Solve the optimality equations using the value iteration algorithm, starting with $V(s) = 0$ for $s = 0, 1$ and 2. You may use a programming environment, but the problem can be solved in a spreadsheet. Run the algorithm for 20 iterations. Plot $V^n(s)$ for $s = 0, 1, 2$, and give the optimal action for each state at each iteration.

(e) Give a bound on the value function after each iteration.

Sequential decision analytics and modeling

These exercises are drawn from the online book *Sequential Decision Analytics and Modeling* available at `http://tinyurl.com/sdaexamplesprint`.

14.29 We are going to perform experiments for an energy storage problem that we can solve exactly using backward dynamic programming. Download the code "EnergyStorage_I" from `http://tinyurl.com/sdagithub`.

a) Using the Python implementation of the basic model, run a grid search for the parameter vector $\theta = (\theta^{buy}, \theta^{sell})$ by varying θ^{sell} over the range from \$20 to \$60 in increments of \$1 for prices, and varying θ^{buy} over the range from \$20 to θ^{sell}, also in increments of \$1. Assume that the price process evolves according to

$$p_{t+1} = \min\{100, \max\{0, p_t + \varepsilon_{t+1}\}\}$$

where ε_{t+1} follows a discrete uniform distribution given by

$$\varepsilon_{t+1} = \begin{cases} -2 & \text{with prob. } 1/5 \\ -1 & \text{with prob. } 1/5 \\ 0 & \text{with prob. } 1/5 \\ +1 & \text{with prob. } 1/5 \\ +2 & \text{with prob. } 1/5 \end{cases}$$

Assume that $p_0 = \$50$.

b) Now solve for an optimal policy by using the backward dynamic programming strategy in section 14.3 of the text (the algorithm has already been implemented in the Python module).

i) Run the algorithm where prices are discretized in increments of \$1, then \$0.50 and finally \$0.25. Compute the size of the state space for each of the three levels of discretization, and plot the run times against the size of the state space.

ii) Using the optimal value function for the discretization of \$1, compare the performance against the best buy-sell policy you found in part (a).

c) Repeat (b), but now assume that the price process evolves according to

$$p_{t+1} = .5p_t + .5p_{t-1} + \varepsilon_{t+1}$$

where ε_{t+1} follows the distribution in part (1). You have to modify the code to handle an extra dimension of the state variable. Compare the run times using the price models assumed in part (a) and part (b) using the single discretization of \$1.

d) Section 8.3.1 of the sequential decision analytics notes introduces a time series model where

$$p_{t+1} = \bar{\theta}_{t0}p_t + \bar{\theta}_{t1}p_{t-1} + \bar{\theta}_{t2}p_{t-2} + \varepsilon_{t+1}. \quad (14.90)$$

The section also provides the updating equations for $\bar{\theta}_t$.

i) For this variation, present the full model of the problem using our canonical framework (states, decisions, exogenous information, transition function, objective function).

ii) How many dimensions does the state variable have? Estimate how long it might take to solve this using Bellman's equation given your experience in parts (b) and (c).

iii) Now consider optimizing the buy-sell policy of part (a). What effect does the more complex price model have on the design of this policy? In particular, how does your policy reflect the value of p_{t-1}?

Diary problem

The diary problem is a single problem you chose (see chapter 1 for guidelines). Answer the following for your diary problem.

14.30 Use your sequential model to write your problem as a dynamic program, and write out Bellman's equation for solving it. Note that you will have to write out the state variables, and then show mathematically how to compute the one-step transition matrix. It is unlikely that you would be able to solve this, so discuss the computational complexity of each of the elements that you would need to solve Bellman's equation. Note that if you have continuous elements in your state variable, you just have to treat the transition matrix as a function that you integrate over, rather than using discrete sums.

Bibliography

Bellman, R. E. (1957), *Dynamic Programming*, Princeton University Press, Princeton, N.J.

Bellman, R. E. (1971), *Introduction to the Mathematical Theory of Control Processes, Vol. II*, Academic Press, New York.

Denardo, E. V. (1982), *Dynamic Programming*, Prentice-Hall, Englewood Cliffs, NJ.

Derman, C. (1962), 'On sequential decisions and Markov chains', *Management Science* **9**(1), 16–24.

Derman, C. (1970), *Finite State Markovian Decision Processes*, Academic Press, New York.

Dreyfus, S. & Law, A. M. (1977), *The Art and Theory of Dynamic Programming*, Academic Press, New York.

Dynkin, E. B. & Yushkevich, A. A. (1979), 'Controlled Markov processes', *in volume Grundlehren der mathematischen Wissenschaften 235 of A Series of Comprehensive Studies in Mathematics. New York: SpringerVerlag*.

Heyman, D. P. & Sobel, M. (1984), *Stochastic Models in Operations Research, Volume II: Stochastic Optimization*, McGraw Hill, New York.

Howard, R. A. (1960), Dynamic programming and Markov processes, MIT Press, Cambridge, MA.

Lewis, F. L. & Vrabie, D. (2012), *Design Optimal Adaptive Controllers*, 3 edn, John Wiley & Sons, Hoboken, NJ.

Manne, A. S. (1960), 'Linear programming and sequential decisions', *Management Science* **6**(3), 259–267.

Nemhauser, G. L. (1966), *Introduction to dynamic programming*, John Wiley & Sons, New York.

Puterman, M. L. (2005), *Markov Decision Processes*, 2nd edn, John Wiley and Sons, Hoboken, NJ.

Ross, S. M. (1983), 'Introduction to Stochastic Dynamic Programming', *Academic Press, New York*.

White, D. J. (1969), *Dynamic Programming*, Holden-Day, San Francisco.