REINFORCEMENT LEARNING AND STOCHASTIC OPTIMIZATION

A unified framework for sequential decisions

Warren B. Powell

June 30, 2021

WILEY-INTERSCIENCE

A JOHN WILEY & SONS, INC., PUBLICATION

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

Library of Congress Cataloging-in-Publication Data:

Optimization Under Uncertainty: A unified framework
Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

CHAPTER 2

CANONICAL PROBLEMS AND APPLICATIONS

The vast array of sequential decision problems has produced at least 15 distinct communities (which we listed in section 1.2) that have developed methods for modeling and solving these problems. Just as written and spoken languages have evolved from different roots, these communities feature roughly eight fundamentally different notational systems, in addition to what could be called dialects, with notation derived from one of the core systems.

Hidden in these different notational "languages" are methods that are sometimes truly original, while others are creative evolutions, and yet others are simply the same method with a different name. Motivating the different methods are the classes of problems that have caught the imagination of each community. Not surprisingly, individual research communities steadily move into new problems, which then motivate new methods.

This chapter provides, in section 2.1, an overview of these different communities and their modeling style. This chapter provides very brief introductions to the most important canonical models of each community, in the notation of that community. In some cases we pause to hint at how we would take a different perspective. Then, section 2.2 summarizes the universal modeling framework that we will use in this book, which can be used to model each of the canonical problems in section 2.1. Then, section 2.3 provides a short summary of different application settings.

2.1 CANONICAL PROBLEMS

Each community in stochastic optimization has a canonical modeling framework that they use to illustrate their problem domain. Often, these canonical problems lend themselves

to an elegant solution technique which then becomes a hammer looking for a nail. While these tools are typically limited to a specific problem class, they often illustrate important ideas that become the foundation of powerful approximation methods. For this reason, understanding these canonical problems helps to provide an important foundation for stochastic optimization.

For a reader new to all of these fields, these canonical problems can just be skimmed the first time through the book. It is important to realize that every one of these fields studies a form of sequential decision problem which can be modeled with the universal modeling framework that we first introduced in section 1.3, and then present in 2.2 in more detail.

2.1.1 Stochastic search

As we are going to learn, if there is a single problem that serves as a single umbrella for almost all stochastic optimization problems (at least, all the ones that use an expectation), it is a problem that is often referred to as stochastic search, which is written

$$\max_x \mathbb{E}F(x,W), \tag{2.1}$$

where x is a deterministic variable, or a vector (or, as we will show, a function). The expectation is over the random variable W, which can be a vector, as well as a sequence of random variables $W_1,\ldots,W_t,\ldots,W_T$ that evolve over time. We are going to refer to the notational style used in the expectation, where we do not indicate what we are taking the expectation over, in equation (2.1) as the *compact form* of the expectation.

We prefer the style where we make the dependence on the random variable explicit by writing

$$\max_x \mathbb{E}_W F(x,W). \tag{2.2}$$

We refer to the style used in equation (2.2), where we indicate what random variable we are taking the expectation over, as the *expanded form* of the expectation. While probabilists frown on this habit, any notation that improves clarity should be encouraged. We are also going to introduce problems where it is useful to express the dependence on an initial state variable S^0, which is done by writing

$$\max_x \mathbb{E}\{F(x,W)|S^0\} \quad = \quad \mathbb{E}_{S^0}\mathbb{E}_{W|S^0}F(x,W). \tag{2.3}$$

Initial state variables can express the dependence of the problem on either deterministic or probabilistic information (say, a distribution about an unknown parameter). For example, we might assume that W is normally distributed with mean μ, where μ is also uncertain (it might be uniformly distributed between 0 and 10). In this case, the first expectation in (2.3), $\mathbb{E}_{S^0}$, is over the uniform distribution for μ, while the second expectation, $\mathbb{E}_{W|S^0}$, is over the normal distribution for W given a value for the mean μ. We see that the form in equation (2.3) does a better job of communicating the uncertainties involved.

There are problems where the initial state S^0 may change each time we solve a problem. For example, S^0 might capture the medical history of a patient, after which we have to decide on a course of treatment, and then we observe medical outcomes. We will sometimes use the style in (2.1) for compactness, but we are going to use the expanded form in (2.3) as our default style (the motivation for this becomes more apparent when you start working on real applications).

There are two lines of research addressing the solution of (2.3) (or (2.1)), both dating to 1951. The first and best known is the research based on derivative-based stochastic search, which exploits the ability in many applications to compute the gradient of a sample of the function which we would write as

$$\nabla_x F(x, W(\omega))$$

where $W(\omega)$ represents a sample of the random variable W, thereby avoiding the need to compute the expectation (think of this as computing the gradient *after* we learn the outcome of W). We cover this line of research in chapters 5 and 6. There is an extensive literature studying the asymptotic behavior of algorithms (we present our own samples of convergence proofs at the end of chapter 5), but the emphasis of this book will always be on the analysis of algorithms that run in a finite budget (as would always be the case in practice).

The second line of research is based on derivative-free stochastic search, where we no longer assume that we can compute the gradient (even for a sample). Needless to say, this arises in many situations, including any physical experiment where we can sample a process, but not its derivative. Reflecting the breadth of applications, this line of research has spawned work coming from multiple communities, using different solution approaches, which we review in chapter 7.

This basic problem class comes in a number of flavors, depending on the following:

- Initial state S^0 - The initial state will include any deterministic parameters, as well as initial distributions of uncertain parameters. S^0 might be a fixed set of deterministic parameters (such as the temperature at which water boils), or it might change each time we solve our problem (it might include temperature and humidity in a lab), and it might include a probability distribution describing an unknown parameter (such as how a market responds to price).

- Decision x - x can be binary, discrete (and finite, and not too large), categorical (finite, but a potentially very large number of choices), continuous (scalar or vector), or a discrete vector.

- Random information W - The distribution of W may be known or unknown, and the distribution can take on a variety of distributions ranging from nice distributions such as the normal or exponential, or one with heavy tails, spikes, and rare events. W may be a single variable or vector that is realized all at once, or it can be a sequence of variables (or vectors) $W_1, \ldots, W_t, \ldots, W_T$.

- The function $F(x, W)$ may be characterized along several dimensions:

  - Final-reward vs. cumulative reward - Equation (2.1) is the final-reward version, which means that we only care about how well a solution x performs at the end of the search. Often, we are going to need to search for the best answer, and we may care about how well we do while we are searching, in which case we obtain the *cumulative reward* formulation.
  - The noise level (and the nature of the noise) - There are applications where the noise in a function evaluation is minimal (or nonexistent), and others where the noise level is exceptionally high.
  - The cost of a function evaluation - The function $F(x, W)$ may be easy to evaluate (fractions of a second to seconds), or more expensive (minutes to hours to days to weeks).

– Search budget - May be finite (for example, we are limited to N evaluations of the function or its gradient), or infinite (obviously this is purely for analysis purposes - real budgets are always finite). There are even problems where a rule determines when we stop, which may be exogenous or dependent on what we have learned (these are called *anytime problems*).

Problem (2.1) is the asymptotic form of the basic stochastic optimization problem where we seek an optimal solution x^* that is deterministic (there is exactly one of them). Most of this book will focus on the more practical finite budget versions where we run an algorithm (that we call π for reasons that become clear later), for N iterations, to produce a solution $x^{\pi,N}$ which is a random variable, since it depends on the observations of W along the way.

There are two flavors of this problem. The first is what we call the *final reward* formulation, where we are going to run our algorithm π, but we only care about the performance of our final solution $x^{\pi,N}$ at the end of the search. The second is called the *cumulative reward* formulation, which often arises when we are learning in the field, and we have to live with the performance of each intermediate solution $x^{\pi,n}$ for $n = 0, 1, \ldots, N$ while we are learning. We will turn to these two objectives throughout the book, so we are going to take a minute to describe each of them in a bit more detail right now.

Final reward formulation Let S^n represent what we know about the function $F(x) = \mathbb{E}_W F(x, W)$ after n iterations, and let $X^\pi(S^n)$ be the rule we use to choose the next point $x^n = X^\pi(S^n)$ to test, after which we observe W^{n+1} (we may just be able to observe $\hat{F}^{n+1} = F(x^n, W^{n+1})$). In this setting we typically think of the rule $X^\pi(S^n)$ as an algorithm, but we are also going to argue that our search process is a form of sequential decision problem, in which case we refer to $X^\pi(S^n)$ as a *policy*, which is a method for making a decision (in this case, x^n).

Imagine now that we fix our "policy" $X^\pi(S^n)$, and then use this to observe $W^1, W^2, \ldots, W^N$, producing a final decision $x^{\pi,N}$. For example, we might be looking to find the best capacity for a battery or transmission line, the capacity of a dam, or even the strength of an airplane wing, all of which represent decisions that have to be made before any information becomes known. Since $x^{\pi,N}$ depends on our realizations of $W^1, W^2, \ldots, W^N$, it is a random variable. After we find $x^{\pi,N}$, we then have to test it through more observations of W; we let $\widehat{W}$ be the random variable used for testing our solution.

This problem requires that we recognize possibly three types of random variables:

- The initial state S^0, which may include distributions about uncertain parameters.
- The sequence of observations $W^1, \ldots, W^N$ which, combined with our policy $X^\pi(S^n)$, produces the implementation decision $x^{\pi,N}$ which is also a random variable since it depends on the sequence $W^1, \ldots, W^N$.
- The uncertainty $\widehat{W}$ when evaluating the function given $x^{\pi,N}$.

Recognizing all these sources of uncertainty means that we can expand the expectation in (2.3) to obtain the objective function

$$\max_\pi \mathbb{E}_{S^0} E_{W^1,\ldots,W^N|S^0} E_{\widehat{W}|S^0,x^{\pi,N}} F(x^{\pi,N}, \widehat{W}). \tag{2.4}$$

We refer to (2.4) as the *final reward* version of our basic stochastic optimization problem since we are only evaluating the final implementation decision $x^{\pi,N}$ (this is sometimes referred to as *terminal reward*). For final reward problems, the "policy" is often referred to

as an "algorithm" which sequentially searches for the best estimate of the optimal solution x^* to $\max_x \mathbb{E}F(x,W)$. We will return to this problem below, and then in much more depth in chapters 5 and 7.

The classical literature in stochastic search, whether it is derivative-based or derivative-free, uses the final-reward formulation, but the formulation of these problems for a finite search budget as we have in equation (2.4) is unusual. Although in practice we always want the best solution, writing this in the form of searching for an optimal algorithm is not done.

Cumulative reward formulation Now assume that we want to find a policy that maximizes our total reward over the N iterations, which we write as

$$\max_{\pi} \mathbb{E}_{S^0}\mathbb{E}_{W^1,\ldots,W^N|S^0} \sum_{n=0}^{N-1} F(X^\pi(S^n),W^{n+1}). \tag{2.5}$$

We refer to (2.5) as the *cumulative reward* version of our elementary stochastic optimization problem since we add up our performance over the n iterations of our search.

Cumulative reward problems arise when we have to learn in the field, which means that we have to live with the results of each experiment. For example, imagine that x^n is how much inventory we choose to stock, and that $W+1$ is the sales and $F(x^n,W^{n+1})$ is our profit. Since these are actual profits we make each day, we want to maximize total profits as we are learning.

The cumulative-reward formulation arises in many real applications, but it was not until the multiarmed bandit problem emerged (see section 2.1.9 below) that people started considering the cumulative-reward form of the stochastic search problem. By contrast, the cumulative reward objective is standard in dynamic programming, as we will see in section 2.1.3 for Markov decision processes.

2.1.2 Decision trees

Decision trees are easily one of the most familiar ways to depict sequential decision problems, with or without uncertainty. Figure 2.1 illustrates a simple problem of determining whether to hold or sell an asset. If we decide to hold, we observe changes in the price of the asset and then get to make the decision of holding or selling.

Figure 2.1 illustrates the basic elements of a decision tree. Square nodes represent points where decisions are made, while circles represent points where random information is revealed. We solve the decision tree by rolling backward, calculating the value of being at each node. At an outcome node, we average across all the downstream nodes (since we do not control which node we transition to), while at decision nodes, we pick the best decision based on the one-period reward plus the downstream value.

Almost any dynamic program with discrete states and actions can be modeled as a decision tree. The problem is that decision trees grow explosively, even for relatively small problems. Imagine a setting where there are three decisions (buy, sell, hold an asset), and three random outcomes (say the price changes by +1, -1 or 0). Each sequence of price change followed by decision grows the tree by a factor of 9. Now imagine a trading problem where we get to make decisions once each minute. After just one hour, our tree has grown to $9^{60} \approx 10^{57}$ branches!

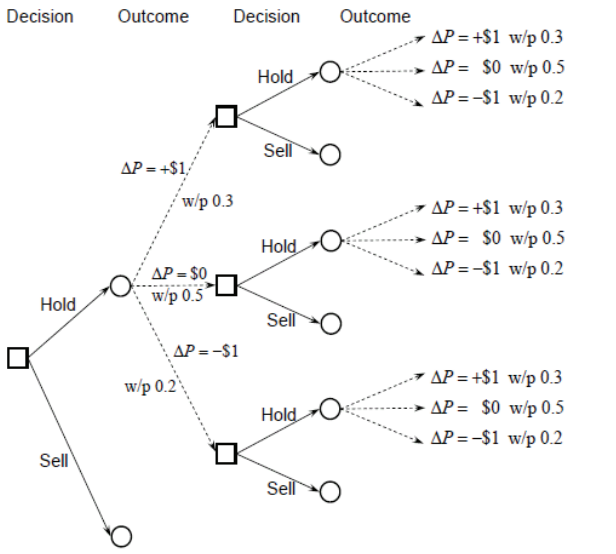

Figure 2.1 Decision tree illustrating the sequence of decisions (hold or sell an asset) and new information (price changes).

2.1.3 Markov decision processes

A Markov decision process is typically modeling using a very standard framework:

State space - $\mathcal{S} = \{s_1, \ldots, s_{|\mathcal{S}|}$ is the set of (discrete states) the system may occupy.

Action space - $\mathcal{A}_s = \{a_1, \ldots a_M\}$ is the set of actions we can take when we are in state s.

Transition matrix - We assume we are given the one-step transition matrix with element

$P(s'|s, a) =$ the probability that state $S_{t+1} = s'$ given that we are in state $S_t = s$ and take action a.

Reward function - Let $r(s, a)$ be the reward we receive when we take action a when we are in state s.

Note that this is modeled without indexing time, since the standard canonical model is for a problem in steady state. Some authors also include a set of "decision epochs" which are the points in time, typically modeled as $t = 1, 2, \ldots$, when we compute our state variable and choose an action.

For example, if we have $s \in \mathcal{S}$ units of inventory, purchase $a \in \mathcal{A}_s$ more units, and then sell a random quantity $\hat{D}$ which produces updated inventory

$$s' = \max\{0, s + a - \hat{D}\},$$

then our one-step transition matrix would be computed from

$$P(s'|s, a) = Prob[\hat{D} = \max\{0, (s + a) - s'\}].$$

Our reward function might be

$$r(s,a) = p\min\{s+a, \hat{D}\} - ca,$$

where c is the unit cost of purchasing an item of inventory and p is our sales price for meeting as much demand as we can.

If we are solving a finite horizon problem, let $V_t(S_t)$ be the optimal value of being in state S_t and behaving optimally from time t onward. If we are given $V_{t+1}(S_{t+1})$, we can compute $V_t(S_t)$ using

$$V_t(S_t) = \max_{a\in\mathcal{A}_s}\left(r(S_t,a) + \gamma\sum_{s'\in\mathcal{S}} P(s'|S_t,a)V_{t+1}(S_{t+1}=s')\right). \tag{2.6}$$

where γ is a discount factor (presumably to capture the time value of money). Note that to compute $V_t(S_t)$, we have to loop over every possible value of $S_t \in \mathcal{S}$ and then solve the maximization problem.

Equation (2.6) may seem somewhat obvious, but when first introduced it was actually quite a breakthrough, and is known as *Bellman's optimality equation* in operations research and computer science, or *Hamilton-Jacobi equations* in control theory (although this community typically writes it for continuous states and actions/controls).

Equation (2.6) is the foundation for a major class of policies that we refer to as policies based on value function (or VFA policies). Specifically, if we know $V_{t+1}(S_{t+1})$, then we would make a decision at time t when we are in state S_t by solving

$$X_t^\pi(S_t) = \arg\max_{a\in\mathcal{A}_s}\big(r(S_t,a) + \gamma\sum_{s'\in\mathcal{S}} P(s'|S_t,a)V_{t+1}(S_{t+1}=s')\big).$$

If we can compute the value functions exactly using equation (2.6), then this is a rare instance of an optimal policy.

If the one-step transition matrix $P(s'|S_t,a)$ can be computed (and stored), then equation (2.6) is quite easy to compute starting at time T (when we assume $V_T(S_T)$ is given, where it is fairly common to use $V_T(S_T)=0$) and progressing backward in time.

There has been considerable interest in this community on steady state problems, where we assume that as $t\rightarrow\infty$, that $V_t(S_t)\rightarrow V(S)$. In this case, (2.6) becomes

$$V(s) = \max_{a\in\mathcal{A}_s}\left(r(s,a) + \gamma\sum_{s'\in\mathcal{S}} P(s'|s,a)V(s')\right). \tag{2.7}$$

Now we have a system of equations that we have to solve to find $V(s)$. We review these methods in some depth in chapter 14.

Bellman's equation was viewed as a major computational breakthrough when it was first introduced, because it avoids the explosion of decision trees. However, people (including Bellman) quickly realized that there was a problem when the state s is a vector (even if it is still discrete). The size of the state space grows exponentially with the number of dimensions, typically limiting this method to problems where the state variable has at most three or four dimensions. This is widely known as the "curse of dimensionality."

Bellman's equation actually suffers from three curses of dimensionality. In addition to the state variable, the random information (buried in the one-step transition $P(s'|s,a)$) might also be a vector. Finally, the action a might be a vector x. It is common for people to dismiss "dynamic programming" (but they mean discrete Markov decision processes)

because of "the curse of dimensionality" (they could say because of "the curses of dimensionality"), but the real issue is the use of lookup tables. There are strategies for overcoming the curses of dimensionality, but if it were easy, this would be a much shorter book.

2.1.4 Optimal control

The optimal control community is most familiar with the deterministic form of a control problem, which is typically written in terms of the "system model" (transition function)

$$x_{t+1} = f(x_t, u_t),$$

where x_t is the state variable and u_t is the control (or action, or decision). A typical engineering control problem might involve controlling a rocket (think of getting SpaceX to land after its takeoff), where the state x_t is the location and velocity of the rocket (each in three dimensions), while the control u_t would be the forces in all three dimensions on the rocket. How the forces affect the location and speed of the rocket (that is, its state x_t) is all contained in the transition function $f(x_t, u_t)$.

The transition function $f(x_t, u_t)$ is a particularly powerful piece of notation that we will use throughout the book (we write the transition as $S_{t+1} = S^M(S_t, x_t, W_{t+1})$). It captures the effect of a decision x_t (such as move to a location, add inventory, use a medical treatment, or apply force to a vehicle) on the state x_t.

The problem is to find u_t that solves

$$\min_{u_0,\ldots,u_T} \sum_{t=0}^{T} L(x_t, u_t) + J_T(x_T), \tag{2.8}$$

where $L(x, u)$ is a "loss function" and $J_T(x_T)$ is a terminal cost. Equation (2.8) can be stated recursively using

$$J_t(x_t) = \max_{u_t} \big(L(x_t, u_t) + J_{t+1}(x_{t+1})\big) \tag{2.9}$$

where $x_{t+1} = f(x_t, u_t)$. Here, $J_t(x_t)$ is known as the "cost-to-go" function, which is simply different notation for the value function $V_t(S_t)$ in section 2.1.3.

A solution strategy which is so standard that it is often stated as part of the model is to view the transition $x_{t+1} = f(x_t, u_t)$ as a constraint that can be relaxed, producing the objective

$$\min_{u_0,\ldots,u_T} \sum_{t=0}^{T} \big(L(x_t, u_t) + \lambda_t(x_{t+1} - f(x_t, u_t))\big) + J_T(x_T), \tag{2.10}$$

where λ_t is a set of Lagrange multipliers known as "co-state variables." The function

$$H(x_0, u) = \sum_{t=0}^{T} \big(L(x_t, u_t) + \lambda_t(x_{t+1} - f(x_t, u_t))\big) + J_T(x_T)$$

is known as the Hamiltonian.

A common form for the objective in (2.8) is an objective function that is quadratic in the state x_t and control u_t, given by

$$\min_{u_0,\ldots,u_T} \sum_{t=0}^{T} \big((x_t)^T Q_t x_t + (u_t)^T R_t u_t\big), \tag{2.11}$$

Although it takes quite a bit of algebra, it is possible to show that the optimal solution to (2.15) can be written in the form of a function $U^{\pi}(x_t)$ which has the form

$$U^*(x_t) = K_t x_t, \tag{2.12}$$

where K_t is a suitably dimensioned matrix that depends on the matrices $(Q_{t'}, R_{t'}), t' \leq t$.

A limitation of this theory is that it is easy to break. For example, simply adding a nonnegativity constraint $u_t \geq 0$ invalidates this result. The same is true if we make any changes to the objective function, and there are a *lot* of problems where the objective is not quadratic in the state variables and decision variables.

There are many problems where we need to model uncertainty in how our process evolves over time. The most common way to introduce uncertainty is through the transition function, which is typically written as

$$x_{t+1} = f(x_t, u_t, w_t) \tag{2.13}$$

where w_t is random at time t (this is standard notation in the optimal control literature, where it is common to model problems in continuous time). w_t might represent random demands in an inventory system, the random cost when traversing from one location to another, or the noise when measuring the presence of disease in a population. Often, w_t is modeled as additive noise which would be written

$$x_{t+1} = f(x_t, u_t) + w_t, \tag{2.14}$$

where w_t might be thought of as the wind pushing our rocket off course.

When we introduce noise, it is common to write the optimization problem as

$$\min_{u_0,\ldots,u_T} \mathbb{E} \sum_{t=0}^{T} \big((x_t)^T Q_t x_t + (u_t)^T R_t u_t\big). \tag{2.15}$$

The problem with this formulation is that we have to recognize that the control u_t at time t is a random variable that depends on the state x_t, which in turn depends on the noise terms $w_0, \ldots, w_{t-1}$.

To convert our original deterministic control problem to a stochastic control problem, we just have to follow the guidance we provided in section 1.6.3. We begin by introducing a *control law* (in the language of optimal control) that we represent by $U^{\pi}(x_t)$ (we refer to this as a policy). Now our problem is to find the best policy ("control law") that solves

$$\min_{\pi} \mathbb{E}_{w_0,\ldots,w_T} \sum_{t=0}^{T} \big((x_t)^T Q_t x_t + (U_t^{\pi}(x_t))^T R_t U_t^{\pi}(x_t)\big). \tag{2.16}$$

A significant part of this book is focused on describing methods for finding good policies. We revisit optimal control problems in section 14.11.

The language of optimal control is widely used in engineering (mostly for deterministic problems) as well as finance, but is otherwise limited to these communities. However, it is the notation of optimal control that will form the foundation of our own modeling framework.

2.1.5 Approximate dynamic programming

The core idea of approximate dynamic programming is to use machine learning methods to replace the value function $V_t(S_t)$ (see equation (2.6)) with an approximation $\overline{V}_t^n(S_t|\theta)$

(assume this is after n iterations). We can use any of a variety of approximation strategies that we cover in chapter 3. Let a_t be our decision at time t (such as how much inventory to order or what drug to prescribe). Let $\bar{\theta}^n$ be our estimate of θ after n updates. Assuming we are in a state S_t^n (this might be our inventory at time t during iteration n), we could use this approximation to create a sampled observation of the value of being in state S_t^n using

$$\hat{v}_t^n = \max_{a_t} \big(C(S_t^n, a_t) + \overline{V}_{t+1}(S_{t+1}^n|\bar{\theta}^{n-1})\big),$$

where $\bar{\theta}^{n-1}$ is our estimate of θ after $n-1$ iterations.

We can then use $\hat{v}_t^n$ to update our estimate $\bar{\theta}_t^n$ to obtain $\bar{\theta}_t^{n+1}$. How this is done depends on how we are approximating $\overline{V}_t^n(S_t|\theta)$ (a variety of methods are described in chapter 3). There are, in addition, other ways to obtain the sampled observation $\hat{v}_t^n$.

Given a value function approximation $\overline{V}_{t+1}(S_{t+1}^n|\bar{\theta}^{n-1})$, we have a method for making decisions (that is, a policy) using

$$A^\pi(S_t^n) = \arg\max_{a_t} \big(C(S_t^n, a_t) + \overline{V}_{t+1}(S_{t+1}^n|\bar{\theta}^{n-1})\big),$$

where "$\arg\max_a$ returns the value of a that maximizes the expression. This is what we will be calling a "VFA-based policy."

The idea of using approximate value functions started with Bellman in 1959, but was then independently re-invented in the optimal control community (which used neural networks to approximate continuous value functions) in the 1970s, and computer science, where it became known as reinforcement learning, in the 1980s and 1990s. These methods are covered in considerably more depth in chapters 16 and 17. It has also been adapted to stochastic resource allocation problems where methods have been developed that exploit concavity (when maximizing) of the value function (see chapter 18).

2.1.6 Reinforcement learning

While the controls community was developing methods for approximating value functions using neural networks, two computer scientists, Andy Barto and his student Richard Sutton, were trying to model animal behavior, as with mice trying to find their way through a maze to a reward (see figure 2.2). Successes were learned over time by capturing the probability that a path from a particular point in the maze eventually leads to a success.

The basic idea closely paralleled the methods of approximate dynamic programming, but it evolved on its own, with its own style. Instead of learning the value $V(s)$ of being in a state s, the core algorithmic strategy of reinforcement learning involves learning the value $Q(s,a)$ of being in a state s and then taking an action a. The basic algorithm, known as Q-learning, proceeds by computing

$$\hat{q}^n(s^n, a^n) = r(s^n, a^n) + \lambda \max_{a'} \bar{Q}^{n-1}(s', a'), \tag{2.17}$$

$$\bar{Q}^n(s^n, a^n) = (1 - \alpha_{n-1})\bar{Q}^{n-1}(s^n, a^n) + \alpha_{n-1}\hat{q}^n(s^n, a^n). \tag{2.18}$$

Here, λ is a discount factor, but it is different than the discount factor γ that we use elsewhere (occasionally) when solving dynamic problems (see, for example, equation (2.6)). The parameter λ is what could be called an "algorithmic discount factor" since it helps to "discount" the effect of making mistakes in the future that have the effect of reducing (incorrectly) the value of being in state s^n and taking action a^n.

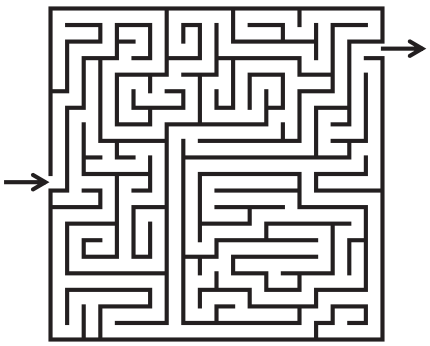

Figure 2.2 Finding a path through a maze.

The updating equation (2.19) is sometimes written

$$\begin{aligned}\bar{Q}^n(s^n,a^n) &= \bar{Q}^{n-1}(s^n,a^n) + \alpha_{n-1}(\hat{q}^n(s^n,a^n) - \bar{Q}^{n-1}(s^n,a^n)) \\ &= \bar{Q}^{n-1}(s^n,a^n) + \alpha_{n-1}\underbrace{(r(s^n,a^n) + \lambda \max_{a'} \bar{Q}^{n-1}(s',a') - \bar{Q}^{n-1}(s^n,a^n))}_{\delta}.\end{aligned} \tag{2.19}$$

where

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

is known as a "temporal difference" because it is capturing the difference between the current estimate $\bar{Q}^{n-1}(s^n,a^n)$ and the updated estimate $(r(s^n,a^n) + \lambda \max_{a'} \bar{Q}^{n-1}(s',a') - \bar{Q}^{n-1}(s^n,a^n))$ from one iteration to the next. Equation (2.19) is known as *temporal difference learning* which is performed with a fixed policy for choosing states and actions. The algorithm is referred to as "TD(λ)" (reflecting the role of the algorithmic discount factor λ) and the method is called "TD learning." In chapters **??** and **??** this will be known as approximate value iteration.

To compute (2.17), we assume we are given a state s^n, such as the location of the mouse in the maze in figure 2.2. We use some method to choose an action a^n, which produces a reward $r(s^n,a^n)$. Next, we randomly sample a downstream state s' that might result from being in a state s^n and taking action a^n. There are two ways we can do this:

1) Model-free learning - We assume that we have a physical system we can observe, such as a doctor making medical decisions or people choosing products off the internet.

2) Model-based learning - Here we assume that we sample the downstream state from the one-step transition matrix $p(s'|s,a)$. In practice, what we are really doing is simulating the transition function from $s' = S^M(s^n,a^n,W^{n+1})$ where the function $S^M(\cdot)$ (using our notation) is the same as equation (2.13) from optimal control, and W^{n+1} is a random variable that we have to sample from some (known) distribution.

Computer scientists often work on problems where a system is being observed, which means they do not use an explicit model of the transition function.

Once we have our simulated downstream state s', we then find what appears to be the best action a' based on our current estimates $\bar{Q}^{n-1}(s', a')$ (known as "Q-factors"). Finally, we then update the estimates of the value of being in states s^n and action a^n. When this logic is applied to our maze in figure 2.2, the algorithm steadily learns the state/action pairs with the highest probability of finding the exit, but it does require sampling all states and actions often enough.

There are many variations of Q-learning that reflect different rules for choosing the state s^n, choosing the action a^n, what is done with the updated estimate $\hat{q}^n(s^n, a^n)$, and how the estimates $\bar{Q}^n(s, a)$ are calculated. For example, equations (2.17) - (2.19) reflect a lookup table representation, but there is considerable ongoing research where $\bar{Q}(s, a)$ is approximated with a deep neural network.

As readers of this book will see, approximating value functions is not an algorithmic panacea, and as the RL community expanded to a broader range of problems, researchers started introducing different algorithmic strategies, which will emerge in this book as samples from each of four classes of policies (policies based on value function approximations is just one). Today, "reinforcement learning" applies more to a community working on sequential decision problems using a wide range of strategies, which is how it made its way into the title of this book.

2.1.7 Optimal stopping

A classical problem in stochastic optimization is known as the optimal stopping problem. Imagine that we have a stochastic process W_t (this might be prices of an asset) which determines a reward $f(W_t)$ if we stop at time t (the price we receive if we stop and sell the asset). Let $\omega \in \Omega$ be a sample path of $W_1, \ldots, W_T$ (we are going to limit our discussion to finite horizon problems, which might represent a maturation date on a financial option). Let

$$X_t(\omega) = \begin{cases} 1 & \text{if we stop at time } t, \\ 0 & \text{otherwise.} \end{cases}$$

Let τ be the time t when $X_t = 1$ (we assume that $X_t = 0$ for $t > \tau$). This notation creates a problem, because ω specifies the entire sample path, which seems to suggest that we are allowed to look into the future before making our decision at time t. Don't laugh - this mistake is easy to make when backtesting policies using historical data. Furthermore, it is actually a fairly standard approximation in the field of stochastic programming which we revisit in chapter 19 (in particular, see two-stage stochastic programming in section 19.9).

To fix this, we require that the function X_t be constructed so that it depends only on the history $W_1, \ldots, W_t$. When this is the case τ is called a *stopping time*. The optimization problem can then be stated as

$$\max_{\tau} \mathbb{E} X_\tau f(W_\tau), \tag{2.20}$$

where we require τ to be a "stopping time." Mathematicians will often express this by requiring that τ (or equivalently, X_t) be an "$\mathcal{F}_t$-measurable function" which is just another way of saying that τ is not computed with information from points in time later than τ.

This language is familiar to students with training in measure-theoretic probability, which is *not* necessary for developing models and algorithms for stochastic optimization. Later, we are going to provide an easy introduction to these ideas (in chapter 9, section 9.13), and then explain why we do not need to use this vocabulary.

More practically, the way we are going to solve the stopping problem in (2.20) is that we are going to create a function $X^\pi(S_t)$ that depends on the state of the system at time t. For example, imagine that we need a policy for selling an asset. Let $R_t = 1$ if we are holding the asset, and 0 otherwise. Assume that $p_1, p_2, \ldots, p_t$ is the history of the price process, where we receive p_t if we sell at time t. Further assume that we create a smoothed process $\bar{p}_t$ using

$$\bar{p}_t = (1 - \alpha)\bar{p}_{t-1} + \alpha p_t.$$

At time t, our state variable is $S_t = (R_t, \bar{p}_t, p_t)$. A sell policy might look like

$$X^\pi(S_t|\theta) = \begin{cases} 1 & \text{If } \bar{p}_t > \theta^{max} \text{ or } \bar{p}_t < \theta^{min}, \\ 0 & \text{Otherwise.} \end{cases}$$

Finding the best policy means finding the best $\theta = (\theta^{min}, \theta^{max})$ by solving

$$\max_\theta \mathbb{E} \sum_{t=0}^{T} p_t X^\pi(S_t|\theta). \tag{2.21}$$

Our stopping time, then, is the earliest time $\tau = t$ where $X^\pi(S_t|\theta) = 1$.

Optimal stopping problems arise in a variety of settings. Some examples include:

European options - A European option on a financial asset gives you the right to sell the asset at a specified date in the future.

American options - An American option gives you the right to sell the asset on or before a specified date.

Machine replacement - While monitoring the status of a (typically complex) piece of machinery, we need to create a policy that tells us when to stop and repair or replace.

Homeland security - The National Security Administration collects information on many people. The NSA needs to determine when to start tracking someone, when to stop (if they feel the target is of no risk) or when to act (when they feel the target is of high risk).

Clinical trials - A drug company running a clinical trial for a drug has to know when to stop the trial and declare success or failure.

Optimal stopping may look like a disarmingly easy problem, given the simplicity of the state variable. However, in real applications there is almost always additional information that needs to be considered. For example, our asset selling problem may depend on a basket of indices or securities that greatly expands the dimensionality of the state variable. The machine replacement problem might involve a number of measurements that are combined to make a decision. The homeland security application could easily involve a number of factors (places the person has visited, the nature of communications, and recent purchases). Finally, health decisions invariably depend on a number of factors that are unique to each patient.

2.1.8 Stochastic programming

Imagine that we have a problem where we have to decide how many Christmas trees to plant which will determine our inventory five years into the future. Call this decision x_0 (we suppress the year we are planning for), which may be a vector determining the plantings at different wholesalers around a region. Then, when we arrive at the sales year (five years later), we see the demand D_1 for Christmas trees and the prices p_1 that the retailers are willing to pay and then have to make shipping decisions x_1 to each retailer.

Let $W_1 = (D_1, p_1)$ represent this random information, and let ω refer to a sample realization of W_1, so that $W_1(\omega) = (D_1(\omega), p_1(\omega))$ is one possible realization of demands and prices. We make the decision x_1 after we see this information, so we have a decision $x_1(\omega)$ of shipping decisions for each possible realization ω of demands for Christmas trees. The stochastic programming community usually refers to each outcome ω as a *scenario*. Assume that c_0 is the costs of producing trees at each location, and c_1 are the costs associated with the vector x_1.

Assume for the moment that $\Omega = (\omega_1, \omega_2, \ldots, \omega_K)$ is a (not too large) set of possible outcomes ("scenarios") for the demand $D_1(\omega)$ and price $p_1(\omega)$. Our second stage decisions $x_1(\omega)$ are constrained by what we planted in the first stage x_0, and we are limited by how much we sell. These two constraints are written as

$$\begin{aligned} A_1 x_1(\omega) &\leq x_0, \\ B_1 x_1(\omega) &\leq D_1(\omega). \end{aligned}$$

Let $\mathcal{X}_1(\omega)$ be the feasible region for $x_1(\omega)$ defined by these constraints. This allows us to write our problem over both stages as

$$\max_{x_0} \left(-c_0 x_0 + \sum_{\omega \in \Omega} p(\omega) \max_{x_1(\omega) \in \mathcal{X}_1(\omega)} (p_1(\omega) - c_1) x_1(\omega) \right). \tag{2.22}$$

In the language of stochastic programming, the second stage decision variables, $x_1(\omega)$, are called "recourse variables" since they represent how we may respond as new information becomes available (which is the definition of "recourse"). Two-stage stochastic programs are basically deterministic optimization problems, but they can be *very large* deterministic optimization problems, albeit ones with special structure.

For example, imagine that we allow the first stage decision x_0 to "see" the information in the second stage, in which case we would write it as $x_0(\omega)$. In this case, we obtain a series of smaller problems, one for each ω. However, now we are allowing x_0 to cheat by seeing into the future. We can overcome this by introducing a *nonanticipativity constraint* which might be written

$$x^0(\omega) - x^0 = 0. \tag{2.23}$$

Now, we have a family of first stage variables $x_0(\omega)$, one for each ω, and then a single variable x_0, where we are trying to force each $x_0(\omega)$ to be the same (at which point we would say that x_0 is "nonanticipative"). Algorithmic specialists can exploit the nonanticipacity constraint (2.23) by relaxing it, then solving a series of smaller problems (perhaps in parallel), and then introducing linking mechanisms so that the overall procedure converges toward a solution that satisfies the nonanticipativity constraint.

We would call the optimization problem in (2.22) (along with the associated constraints for time periods 0 and 1) a stochastic optimization *problem*. In practice, these applications

tend to arise in the context of sequential decision problems, where we would be looking for the best decision x_t at time t that considers the uncertain future (call this $t+1$, although it can be multiple time periods $t+1, \ldots, t+H$), giving us a *policy*

$$X_t^\pi(S_t) = \arg\max_{x_t \in \mathcal{X}_t} \left(-c_t x_t + \sum_{\omega \in \Omega} p_{t+1}(\omega) \max_{x_{t+1}(\omega) \in \mathcal{X}_{t+1}(\omega)} \left((p_{t+1}(\omega) - c_{t+1}) x_{t+1}(\omega) \right) \right). \tag{2.24}$$

The optimization problems in (2.22) and (2.24) are the same, but the goal in solving (2.24) is just to find a decision x_t to implement, after which we roll forward to time $t+1$, update the uncertain future $t+2$, and repeat the process. The decisions $x_{t+1}(\omega)$ for each of the scenarios ω are never actually implemented; we plan them only to help us improve the decision x_t that we are going to implement now. This is a policy for solving an optimization problem which is typically not modeled explicitly. We show how the objective function should be modeled in section 2.2 below.

2.1.9 The multiarmed bandit problem

The classic information acquisition problem is known as the *multiarmed bandit problem* which is a colorful name for our cumulative reward problem introduced in section 2.1.1. This problem has received considerable attention since it was first introduced in the 1950's; the term appears in thousands of papers (per year!).

The bandit story proceeds as follows. Consider the situation faced by a gambler trying to choose which slot machine $x \in \mathcal{X} = \{1, 2, ..., M\}$ to play. Now assume that the winnings may be different for each machine, but the gambler does not know the winning probabilities. The only way to obtain information is to actually play a slot machine. To formulate this problem, let

$$\begin{aligned} x^n &= \begin{cases} 1 & \text{the machine we choose to play next after finishing the } n^{th} \text{ trial,} \\ 0 & \text{otherwise,} \end{cases} \\ W_x^n &= \text{winnings from playing slot machine } x = x^{n-1} \text{ during the } n^{th} \text{ trial.} \end{aligned}$$

We choose what arm to play in the n^{th} trial after finishing the $n-1^{st}$ trial. We let S^n be the belief state after playing n machines. For example, let

$$\begin{aligned} \mu_x &= \text{a random variable giving the true expected winnings from machine } x, \\ \bar{\mu}_x^n &= \text{our estimate of the expected value of } \mu \text{ after } n \text{ trials,} \\ \sigma_x^{2,n} &= \text{the variance of our belief about } \mu_x \text{ after } n \text{ trials.} \end{aligned}$$

Now assume that our belief about μ is normally distributed (after n trials) with mean $\bar{\mu}_x^n$ and variance $\sigma_x^{2,n}$. We can write our belief state as

$$S^n = (\bar{\mu}_x^n, \sigma_x^{2,n})_{x \in \mathcal{X}}.$$

Our challenge is to find a policy $X^\pi(S^n)$ that determines which machine x^n to play for the $n+1^{st}$ trial. We have to find a policy that allows us to better learn the true mean values μ_x, which means we are going to have to sometimes play a machine x^n where the estimated reward $\bar{\mu}_x^n$ is not the highest, but where we acknowledge that this estimate may not be accurate. However, we may end up playing a machine whose average reward μ_x

actually is lower than the best, which means we are likely to incur lower winnings. The problem is to find the policy that maximizes winnings over time.

One way to state this problem is to maximize expected discounted winnings over an infinite horizon

$$\max_{\pi} \mathbb{E} \sum_{n=0}^{\infty} \gamma^n W_{x^n}^{n+1},$$

where $x^n = X^\pi(S^n)$ and where $\gamma < 1$ is a discount factor. Of course, we could also pose this as a finite horizon problem (with or without discounting).

An example of a policy that does quite well is known as the interval estimation policy, given by

$$X^{IE,n}(S^n|\theta^{IE}) = \arg\max_{x \in \mathcal{X}} \left(\bar{\mu}_x^n + \theta^{IE} \bar{\sigma}_x^{2,n} \right),$$

where $\bar{\sigma}_x^{2,n}$ is our estimate of the variance of $\bar{\mu}_x^n$, given by

$$\bar{\sigma}_x^{2,n} = \frac{\sigma_x^{2,n}}{N_x^n}.$$

Our policy is parameterized by θ^{IE} which determines how much weight to put on the uncertainty in the estimate $\bar{\mu}_x^n$. If $\theta^{IE} = 0$, then we have a pure exploitation policy where we are simply choosing the alternative that seems best. As θ^{IE} increases, we put more emphasis on the uncertainty in the estimate. As we are going to see in chapter 7, effective learning policies have to strike a balance between exploring (trying alternatives which are uncertain) and exploiting (doing what appears to be best).

The multiarmed bandit problem is an example of an online learning problem (that is, where we have to learn by doing), where we want to maximize the cumulative rewards. Some examples of these problems are:

EXAMPLE 2.1

Consider someone who has just moved to a new city and who now has to find the best path to work. Let T_p be a random variable giving the time he will experience if he chooses path p from a predefined set of paths $\mathcal{P}$. The only way he can obtain observations of the travel time is to actually travel the path. Of course, he would like to choose the path with the shortest average time, but it may be necessary to try a longer path because it may be that he simply has a poor estimate.

EXAMPLE 2.2

A baseball manager is trying to decide which of four players makes the best designated hitter. The only way to estimate how well they hit is to put them in the batting order as the designated hitter.

EXAMPLE 2.3

A doctor is trying to determine the best blood pressure medication for a patient. Each patient responds differently to each medication, so it is necessary to try a particular

medication for a while, and then switch if the doctor feels that better results can be achieved with a different medication.

Multiarmed bandit problems have a long history as a niche problem in applied probability and statistics (going back to the 1950s), computer science (starting in the mid 1980s), and engineering and the geosciences (starting in the 1990s). The bandit community has broadened to consider a much wider range of problems (for example, x could be continuous and/or a vector), and a growing range of policies. We revisit this important problem class in chapter 7 where we are going to argue that so-called "multiarmed bandit problems" are actually just derivative-free stochastic optimization problems, which can be solved with any of four classes of policies. The difference between bandit problems and the early research into derivative-free stochastic search is that the stochastic search literature did not explicitly recognize the value of active learning: evaluating the function at x just to better learn the approximation, enabling better decisions later.

We note that derivative-free stochastic search is classically approached using a "final reward" objective function (see section 2.1.1), while the multiarmed bandit literature has been centered on cumulative reward objective, but this is not universally true. There is a version of the multiarmed bandit problem known as the "best arm" bandit problem, which uses a final reward objective.

2.1.10 Simulation optimization

The field known as "simulation optimization" evolved originally from within the simulation community which developed Monte Carlo simulation models for simulating complex systems such as manufacturing processes. Early simulation models in the 1960s were quite slow, and they were often used to search over a series of designs, creating an interest in performing these searches efficiently.

Searching over a finite set of alternatives using noisy evaluations is an example of ranking and selection (a form of derivative free stochastic search), but these applications fostered a group of researchers within the simulation community. One of the first methodological innovations from this community was an algorithm called *optimal computing budget allocation*, or OCBA.

Figure 7.15 illustrates a typical version of an OCBA algorithm. The algorithm proceeds by taking an initial sample $N_x^0 = n_0$ of each alternative $x \in \mathcal{X}$, which means we use $B^0 = Mn_0$ experiments from our budget B. Letting $M = |\mathcal{X}|$, we divide the remaining budget of experiments $B - B^0$ into equal increments of size Δ, so that we do $N = (B - Mn_0)\Delta$ iterations.

After n iterations, assume that we have measured alternative x N_x^n times, and let W_x^m be the m^{th} observation of x, for $m = 1, \ldots, N_x^n$. The updated estimate of the value of each alternative x is given by

$$\theta_x^n = \frac{1}{N_x^n} \sum_{m=1}^{N_x^n} W_x^m.$$

Let $x^n = \arg\max \theta_x^n$ be the current best option.

After using $\sum_x N_x^n$ observations from our budget, at each iteration we increase our allowed budget by $B^n = B^{n-1} + \Delta$ until we reach $B^N = B$. After each increment, the

allocation $N_x^n,\ x \in \mathcal{X}$ is recomputed using

$$\frac{N_x^{n+1}}{N_{x'}^{n+1}} = \frac{\hat{\sigma}_x^{2,n}/(\theta_{x^n}^n - \theta_{x'}^n)^2}{\hat{\sigma}_{x'}^{2,n}/(\theta_{x^n}^n - \theta_{x'}^n)^2} \quad x \neq x' \neq x^n, \tag{2.25}$$

$$N_{x^n}^{n+1} = \hat{\sigma}_{x^n}^n \sqrt{\sum_{i=1, i\neq x^n}^{M} \left(\frac{N_x^{n+1}}{\hat{\sigma}_i^n}\right)^2}. \tag{2.26}$$

We use equations (7.84)-(7.85) to produce an allocation N_x^n such that $\sum_x N_x^n = B^n$. Note that after increasing the budget, it is not guaranteed that $N_x^n \geq N_x^{n-1}$ for some x. If this is the case, we would not measure these alternatives at all in the next iteration. We can solve these equations by writing each N_x^n in terms of some fixed alternative (other than x^n), such as N_1^n (assuming $x^n \neq 1$). After writing N_x^n as a function of N_1^n for all x, we then determine N_1^n so that $\sum N_x^n \approx B^n$ (within rounding).

The complete algorithm is summarized in figure 7.15.

For a number of years, OCBA was closely associated with "simulation optimization," but the community continued to evolve, tackling a wider range of problems and creating methods to meet new challenges. Inevitably there was also some cross-over from other communities. However, similar to other communities, the scope of activities under the umbrella of "simulation optimization" has continued to broaden, encompassing other results from stochastic search (both derivative-free and derivative-based), as well as the tools for sequential decision problems such as approximate dynamic programming and reinforcement learning.

2.1.11 Active learning

Classical (batch) machine learning addresses the problem of fitting a model $f(x|\theta)$ given a dataset $(x^n, y^n),\ n = 1, \ldots, N$ to minimize some error (or loss) function $L(x, y)$. Online learning addresses the setting of fitting the model as the data is arriving in a stream. Given an estimate $\bar{\theta}^n$ based on the first n datapoints, find $\bar{\theta}^{n+1}$ given (x^{n+1}, y^{n+1}). We assume we have no control over the inputs x^n.

Active learning arises when we have partial or complete control over the inputs x^n. It might be a price, size or concentration that we completely control. Or, we might have partial control, as occurs when we choose a treatment for a patient, but cannot control the attributes of the patient.

There are many approaches to classical learning, but a popular one is to make choices where there is the greatest uncertainty. For example, imagine that we have binary outcomes (a customer does or does not purchase a product at a price x). Let x be the attributes of the customer, and let $\bar{p}(x)$ be the probability that this customer will purchase the product. We know the attributes of the customer from their login credentials. The variance of the response is given by $\bar{p}^n(x)(1 - \bar{p}^n(x))$. To minimize the variance, we would want to make an offer to a customer with attribute x that has the greatest uncertainty given by the variance $\bar{p}^n(x)(1 - \bar{p}^n(x))$. This means we would choose the x that solves

$$\max_x \bar{p}^n(x)(1 - \bar{p}^n(x)).$$

This is a very simple example of active learning.

Step 0. Initialization:

Step 0a. Given a computing budget B, let n^0 be the initial sample size for each of the $M = |\mathcal{X}|$ alternatives. Divide the remaining budget $T - Mn_0$ into increments so that $N = (T - Mn_0)/\delta$ is an integer.

Step 0b. Obtain samples W_x^m, $m = 1, \ldots, n_0$ samples of each $x \in \mathcal{X}$.

Step 0c. Initialize $N_x^1 = n_0$ for all $x \in \mathcal{X}$.

Step 0d. Initialize $n = 1$.

Step 1. Compute

$$\theta_x^n = \frac{1}{N_x^n} \sum_{m=1}^{N_x^n} W_x^m.$$

Compute the sample variances for each pair using

$$\hat{\sigma}_x^{2,n} = \frac{1}{N_x^n - 1} \sum_{m=1}^{N_x^n} (W_x^m - \theta_x^n)^2 .$$

Step 2. Let $x^n = \arg\max_{x \in \mathcal{X}} \theta_x^n$.

Step 3. Increase the computing budget by Δ and calculate the new allocation $N_1^{n+1}, \ldots, N_M^{n+1}$ so that

$$\begin{aligned} \frac{N_x^{n+1}}{N_{x'}^{n+1}} &= \frac{\hat{\sigma}_x^{2,n}/(\theta_{x^n}^n - \theta_{x'}^n)^2}{\hat{\sigma}_{x'}^{2,n}/(\theta_{x^n}^n - \theta_{x'}^n)^2} \quad x \neq x' \neq x^n, \\ N_{x^n}^{n+1} &= \hat{\sigma}_{x^n}^n \sqrt{\sum_{i=1, i \neq x^n}^{M} \left(\frac{N_x^{n+1}}{\hat{\sigma}_i^n} \right)^2}. \end{aligned}$$

Step 4. Perform $\max\left(N_x^{n+1} - N_x^n, 0\right)$ additional simulations for each alternative x.

Step 5. Set $n = n + 1$. If $\sum_{x \in \mathcal{X}} N_x^n < B$, go to step 1.

Step 6. Return $x^n \arg\max_{x \in \mathcal{X}} \theta_x^n$.

Figure 2.3 Optimal computing budget allocation procedure.

The relationship between bandit problems and active learning is quite close. As of this writing, the term "active learning" has been increasingly replacing the artificial "multiarmed bandit problem."

2.1.12 Chance constrained programming

There are problems where we have to satisfy a constraint that depends on uncertain information at the time we make a decision. For example, we may wish to allocate inventory with the goal that we cover demand 80 percent of the time. Alternatively, we may wish to schedule a flight so that it is on time 90 percent of the time. We can state these problems using the general form

$$\min_x f(x), \tag{2.27}$$

subject to the probabilistic constraint (often referred to as a chance constraint)

$$\mathbb{P}[C(x, W) \geq 0] \quad \leq \quad \alpha, \tag{2.28}$$

where $0 \leq \alpha \leq 1$. The constraint (2.28) is often written in the equivalent form

$$\mathbb{P}[C(x,W) \leq 0] \quad \geq \quad 1-\alpha. \tag{2.29}$$

Here, $C(x,W)$ is the amount that a constraint is violated (if positive). Using our examples, it might be the demand minus the inventory which is the lost demand if positive, or the covered demand if negative. Or, it could be the arrival time of a plane minus the scheduled time, where positive means a late arrival.

Chance constrained programming is a method for handling a particular class of constraints that involve uncertainty, typically in the setting of a static problem: make decision, see information, stop. Chance constrained programs convert these problems into deterministic, nonlinear programs, with the challenge of computing the probabilistic constraint within the search algorithm.

2.1.13 Model predictive control

There are many settings where we need to think about what is going to happen in the future in order to make a decision now. An example most familiar to all of us is the use of navigation systems that plan a path to the destination using estimated travel times on each link of the network. As we progress, these times may change and the path will be updated.

Making decisions now by optimizing (in some way) over the future is known in the optimal control literature as *model predictive control*, because we are using a (typically approximate) model of the future to make a decision now. An example of an MPC policy is

$$\begin{aligned} U^{\pi}(x_t) &= \arg\min_{u_t}\left(L(x_t,u_t) + min_{u_{t+1},\ldots,u_{t+H}}\sum_{t'=t}^{t+H}L(x_{t'},u_{t'})\right) \\ &= \arg\min_{u_t,\ldots,u_{t+H}}\sum_{t'=t}^{t+H}L(x_{t'},u_{t'}). \end{aligned} \tag{2.30}$$

The optimization problem in (2.30) requires a model over the horizon $t,\ldots,t+H$, which means we need to be able to model losses as well as the system dynamics using $x_{t+1} = f(x_t,u_t)$. A slightly more precise name for this might be "model-based predictive control," but "model predictive control" (or MPC, as it is often known) is the term that evolved in the controls community.

Model predictive control is a widely used idea, often under names such as "rolling horizon procedure" or "receding horizon procedure." Model predictive control is most often written using a deterministic model of the future, primarily because most control problems are deterministic. However, the proper use of the term refers to *any* model of the future (even an approximation) that is used to make a decision now. The two-stage stochastic programming model in section 2.1.8 is a form of model predictive control which uses a stochastic model of the future. We could even solve a full dynamic program, which is typically done when we solve an approximate stochastic model of the future. All of these are forms of "model predictive control." In this book we refer to this approach as a class of policy called "direct lookahead approximations" which we cover in chapter 19.

2.1.14 Robust optimization

The term "robust optimization" has been applied to classical stochastic optimization problems (in particular stochastic programming), but in the mid 1990s, it became associated

with problems where we need to make a decision, such as the design of a device or structure, that works under the *worst* possible settings of the uncontrollable parameters. Examples where robust optimization might arise are

---

■ **EXAMPLE 2.4**

A structural engineer has to design a tall building that minimizes cost (which might involve minimizing materials) so that it can withstand the worst storm conditions in terms of wind speed and direction.

■ **EXAMPLE 2.5**

An engineer designing wings for a large passenger jet wishes to minimize the weight of the wing, but the wing still has to withstand the stresses under the worst possible conditions.

The classical notation used in the robust optimization community is to let u be the uncertain parameters, where we assume that w falls within an *uncertainty set* $\mathcal{W}$. The set $\mathcal{W}$ is designed to capture the random outcomes with some level of confidence that we can parameterize with θ, so we are going to write the uncertainty set as $\mathcal{W}(\theta)$.

The robust optimization problem is stated as

$$\min_{x \in \mathcal{X}} \max_{w \in \mathcal{W}(\theta)} F(x, w). \tag{2.31}$$

Creating the uncertainty set $\mathcal{W}(\theta)$ can be a difficult challenge. For example, if w is a vector with element w_i, one way to formulate $\mathcal{W}(\theta)$ is the box:

$$\mathcal{W}(\theta) = \{w | \theta_i^{lower} \leq w_i \leq \theta_i^{upper}, \forall i\}.$$

where $\theta = (\theta^{lower}, \theta^{upper})$ are tunable parameters that govern the creation of the uncertainty set.

The problem is that the worst outcome in $\mathcal{W}(\theta)$ is likely to be one of the corners of the box, where all the elements w_i are at their upper or lower bound. In practice, this is likely to be an extremely rare event. A more realistic uncertainty set captures the likelihood that a vector w may happen. There is considerable research in robust optimization focused on creating the uncertainty set $\mathcal{W}(\theta)$.

We note that just as we formulated a two-stage stochastic programming *problem* in equation (2.22), and then pointed out that this was really a lookahead *policy* (see equation (2.24)), our robust optimization *problem* given by (2.31) can be written as a robust optimization *policy* if we write it as

$$X^{RO}(S_t) = \arg \min_{x_t \in \mathcal{X}_t} \max_{w_t \in \mathcal{W}_t(\theta)} F(x_t, w_t). \tag{2.32}$$

A number of papers in the robust optimization literature are doing precisely this: they formulate a robust optimization problem at time t, and then use it to make a decision x_t, after which they step forward, observe new information W_{t+1}, and repeat the process. This means that their robust optimization *problem* is actually a form of lookahead *policy*.

2.2 A CANONICAL MODELING FRAMEWORK FOR SEQUENTIAL DECISION PROBLEMS

Now that we have covered most of the major communities dealing with sequential decisions under uncertainty, it is useful to review the elements of all sequential decision problems. We are going to revisit this topic in considerably greater depth in chapter 9, but this discussion provides an introduction and a chance to compare our framework to those we reviewed above.

Our presentation focuses on sequential decision problems under uncertainty, which means new information arrives after each decision is made, but we can always ignore the new information to create a problem comparable to the deterministic control problem in sectionb 2.1.4. We are going to assume our problem evolves over time, but there are many settings where it is more natural to use a counter (the n^{th} experiment, the n^{th} customer).

2.2.1 Our canonical model for sequential decision problems

These problems consist of the following elements:

The state variable - S_t - This captures all the information we need to model the system from time t onward, which means computing the cost/contribution function, constraints on decisions, and any other variables needed to model the transition of this information over time. The state S_t may consist of the physical resources R_t (such as inventories), other deterministic information I_t (price of a product, weather), and the belief state B_t which captures information about a probability distribution describing parameters or quantities that cannot be directly (and perfectly) observed. It is important to recognize that the state variable, regardless of whether it is describing physical resources, attributes of a system, or the parameters of a probability distribution, is always a form of information.

The decision variable - x_t - Decisions (which might be called actions a_t or controls u_t) represent how we control the process. Decisions are determined by decision functions known as *policies*, also known as control laws in control theory. If our decision is x_t, we will designate our policy by $X^\pi(S_t)$. Similarly, if we wish to use a_t or u_t as our decision variable, we would use $A^\pi(S_t)$ or $U^\pi(S_t)$ as our policy. If $\mathcal{X}_t$ is our feasible region (which depends on information in S_t), we assume that $X^\pi(S_t) \in \mathcal{X}_t$.

Exogenous information - W_{t+1} - This is the information that first becomes known at time $t+1$ from an exogenous source (for example, the demand for product, the speed of the wind, the outcome of a medical treatment, the results of a laboratory experiment). W_{t+1} can be a high dimensional vector of prices (for all the different stocks) or demands for products.

The transition function - This function determines how the system evolves from the state S_t to the state S_{t+1} given the decision that was made at time t and the new information that arrived between t and $t+1$. We designate the transition function (also known as the system model) by

$$S_{t+1} = S^M(S_t, x_t, W_{t+1}).$$

Note that W_{t+1} is a random variable when we make the decision x_t. Throughout, we assume that any variable indexed by t (or n) is known at time t (or after n observations).

The objective function - This function specifies the costs being minimized, the contributions/rewards being maximized, or other performance metrics. Let $C(S_t, x_t)$ be the contribution we are maximizing given the decision x_t, and given the information in S_t which may contain costs, prices and information for constraints. A basic form of objective function might be given by

$$F^{\pi}(S_0) = \mathbb{E}_{S_0}\mathbb{E}_{W_1,\ldots,W_T|S_0}\left\{\sum_{t=0}^{T} C(S_t, X^{\pi}(S_t))\right\}. \tag{2.33}$$

Our goal would be to find the policy that solves

$$\max_{\pi} F^{\pi}(S_0). \tag{2.34}$$

In chapters 7 and 9 we will illustrate a number of other forms of objectives.

If we are using a counter, we would represent the state by S^n, decisions by x^n, and the exogenous information by W^{n+1}. There are some problems where we need to index by both time (such as the hour within a week) and a counter (such as the n^{th} week). We would do this using S_t^n.

We now illustrate this framework using a resource acquisition problem:

Narrative - Our asset acquisition problem involves maintaining an inventory of some resource (cash in a mutual fund, spare engines for aircraft, vaccines, ...) to meet random demands over time. We assume that purchase costs and sales prices also vary over time.

State variables - The state variable is the information we need to make a decision and compute functions that determine how the system evolves into the future. In our asset acquisition problem, we need three pieces of information. The first is R_t, the resources on hand before we make any decisions (including how much of the demand to satisfy). The second is the demand itself, denoted D_t, and the third is the price p_t. We would write our state variable as $S_t = (R_t, D_t, p_t)$.

Decision variables - We have two decisions to make. The first, denoted x_t^D, is how much of the demand D_t during time interval t that should be satisfied using available assets, which means that we require $x_t^D \leq R_t$. The second, denoted x_t^O, is how many new assets should be acquired at time t which can be used to satisfy demands during time interval $t + 1$.

Exogenous information - The exogenous information process consists of three types of information. The first is the new demands that arise between t and $t + 1$, denoted $\hat{D}_{t+1}$. The second is the change between t and $t + 1$ in the price at which we can sell our assets, denoted $\hat{p}_{t+1}$. Finally, we are going to assume that there may be exogenous changes to our available resources. These might be blood donations or cash deposits (producing positive changes), or equipment failures and cash withdrawals (producing negative changes). We denote these changes by $\hat{R}_{t+1}$. We let W_{t+1} represent all the new information that is first learned between t and $t + 1$ (that is, after decision x_t is made), which for our problem would be written $W_{t+1} = (\hat{R}_{t+1}, \hat{D}_{t+1}, \hat{p}_{t+1})$.

In addition to specifying the types of exogenous information, for stochastic models we also have to specify the likelihood of a particular outcome. This might come in

the form of an assumed probability distribution for $\hat{R}_{t+1}$, $\hat{D}_{t+1}$, and $\hat{p}_{t+1}$, or we may depend on an exogenous source for sample realizations (the actual price of the stock or the actual travel time on a path).

Transition function - The evolution of the state variables S_t is described using

$$S_{t+1} = S^M(S_t, x_t, W_{t+1}),$$

where

$$\begin{aligned} R_{t+1} &= R_t - x_t^D + x_t^O + \hat{R}_{t+1}, \\ D_{t+1} &= D_t - x_t^D + \hat{D}_{t+1}, \\ p_{t+1} &= p_t + \hat{p}_{t+1}. \end{aligned}$$

This model assumes that unsatisfied demands are held until the next time period.

Objective function - We compute our contribution $C_t(S_t, x_t)$ which might depend on our current state and the action x_t that we take at time t. For our asset acquisition problem (where the state variable is R_t), the contribution function is

$$C_t(S_t, x_t) = p_t x_t^D - c_t x_t^O.$$

In this particular model, $C_t(S_t, x_t)$ is a deterministic function of the state and action. In other applications, the contribution from action x_t depends on what happens during time $t + 1$.

Our objective function is given by

$$\max_{\pi \in \Pi} \mathbb{E} \left\{ \sum_{t=0}^{T} C_t(S_t, X^\pi(S_t)) | S_0 \right\}.$$

Designing policies will occupy most of the rest of this volume. For an inventory problem such as this, we might use simple rules, or more complex lookahead policies, where we may look into the future with a point forecast, or while capturing the uncertainty of the future.

Chapter 9 is an entire chapter dedicated to filling in details of this basic modeling framework. When modeling a real problem, we encourage readers to describe each of these five elements in this order.

2.2.2 A compact modeling presentation

For readers looking for a more compact way of writing a sequential decision problem that is perhaps more in the style of a classical deterministic math program, we suggest writing it as

$$\max_{\pi} \mathbb{E}_{S_0} \mathbb{E}_{W_1, \ldots, W_T | S_0} \left\{ \sum_{t=0}^{T} C(S_t, X^\pi(S_t)) \right\}, \tag{2.35}$$

where we assume that the policy is designed to satisfy the constraints

$$x_t = X^\pi(S_t) \in \mathcal{X}_t. \tag{2.36}$$

The transition function is given by

$$S_{t+1} = S^M(S_t, X^\pi(S_t), W_{t+1}), \tag{2.37}$$

and where we are given an exogenous information process

$$(S_0, W_1, W_2, \ldots, W_T). \tag{2.38}$$

Of course, this still leaves the problem of describing how to sample the exogenous information process, and how to design the policy. However, we do not feel that we need to say anything about the policy, any more than we need to say something about the decision x in a deterministic math program.

2.2.3 MDP/RL vs. optimal control modeling frameworks

It helps to pause and ask a natural question: of all the fields listed in section 2.1, do any of them match our universal framework? The answer is that there is one that comes close: optimal control (section 2.1.4).

Before describing the strengths of the optimal control modeling framework, we think it helps to start with the modeling framework that has been adopted by the reinforcement learning community, which is the most popular of all of these fields (as of the writing of this book). From its origins in the 1980s, the RL community adopted the modeling framework long used for Markov decision processes, which we presented in section 2.1.3. This framework may be mathematically elegant, but it is extremely clumsy in terms of modeling actual problems. For example, we learn nothing about a problem by defining some "state space" $\mathcal{S}$ or "action space" $\mathcal{A}$. In addition, the one-step transition matrix $P(s'|s,a)$ is almost never computable. Finally, while it is nice to specify the single-period reward function, the real problem is to sum the rewards and optimize over policies.

Now let's contrast this style with that used in optimal control. In this field, we specify state *variables* and decision/control *variables*. It is the field of optimal control that introduced the powerful construct of a transition *function*, which can seem so obvious, and yet is largely ignored by the other communities. The optimal control literature focuses predominantly on deterministic problems, but there are stochastic control problems, most often using the additive noise of equation (2.14).

The optimal control community does not use our standard format of optimizing over policies. Yet, this community has aggressively developed different classes of policies. We observe that the optimal control literature first introduced "linear control laws" (because they are optimal for linear quadratic regulation problems). It was the first to use value function approximations under a variety of names including heuristic dynamic programming, neuro-dynamic programming, and approximate/adaptive dynamic programming. Finally, it introduced (deterministic) lookahead policies (known as "model predictive control"). This spans three of our four classes of policies (PFAs, VFAs and DLAs). We suspect that someone has used the idea of parameterized optimization models for policies (what we call CFAs), but since this strategy has not been recognized as a formal methodology, it is difficult to know if and when it has been first used.

All of the fields in section 2.1 suffer from the habit of tying a modeling framework to a solution approach. Optimal control, along with dynamic programming, assumes that the starting point is Bellman's equation (known as Hamilton-Jacobi equations in the controls community). This is our major point of departure with all of the fields listed above. In our universal modeling framework, none of the five elements provides any indication of how

to design policies. Instead, we end with an objective function (equations (2.33)-(2.34)) where we state that our objective is to find an optimal policy. We defer until later the search over the four classes of policies which we first introduced in section 1.4.1, and will revisit throughout the book.

2.3 APPLICATIONS

We now illustrate our modeling framework using a series of applications. These problems illustrate some of the modeling issues that can arise in actual applications. We often start from a simpler problem, and then show how details can be added. Pay attention to the growth in the dimensionality of the state variable as these complications are introduced.

2.3.1 The newsvendor problems

A popular problem in operations research is known as the newsvendor problem, which is described as the story of deciding how many newspapers to put out for sale to meet an unknown demand. The newsvendor problem arises in many settings where we have to choose a fixed parameter that is then evaluated in a stochastic setting. It often arises as a subproblem in a wide range of resource allocation problems (managing blood inventories, budgeting for emergencies, allocating fleets of vehicles, hiring people). It also arises in other settings, such as bidding a price for a contract (bidding too high means you may lose the contract), or allowing extra time for a trip.

Basic newsvendor - Final reward The basic newsvendor is modeled as

$$F(x,W) = p\min\{x,W\} - cx, \tag{2.39}$$

where x is the number of "newspapers" we have to order before observing our random "demand" W. We sell our newspapers at a price p (the smaller of x and W), but we have to buy all of them at a unit cost c. The goal is to solve the problem

$$\max_x \mathbb{E}_W F(x,W). \tag{2.40}$$

In most cases, the newsvendor problem arises in settings where we can observe W, but we do not known its distribution (this is often referred to as "data driven"). When this is the case, we assume that we have to determine the amount to order x^n at the end of day n, after which we observe demand W^{n+1}, giving us a profit (at the end of day $n+1$) of

$$\hat{F}^{n+1} = F(x^n, W^{n+1}) = p\min\{x^n, W^{n+1}\} - cx^n.$$

After each iteration, we may assume we observe W^{n+1}, although often we only observe $\min(x^n, W^{n+1})$ (which is known as censored observations) or perhaps just the realized profit

$$\hat{F}^{n+1} \quad = \quad p\min\{x^n, W^{n+1}\} - cx^n.$$

We can devise strategies to try to learn the distribution of W, and then use our ability to solve the problem optimally (given in exercise 4.12).

Another approach is to try to learn the function $\mathbb{E}_W F(x,W)$ directly. Either way, let S^n be our belief state (about W, or about $\mathbb{E}_W F(x,W)$) about our unknown quantities. S^n

might be a point estimate, but it is often a probability distribution. For example, we might let $\mu_x = \mathbb{E}F(x,W)$ where we assume that x is discrete (say, the number of newspapers). After n iterations, we might have estimates $\bar{\mu}_x^n$ of $\mathbb{E}F(x,W)$, with standard deviation $\bar{\sigma}_x^n$ where we would then assume that $\mu_x \sim N(\bar{\mu}_x^n, \bar{\sigma}_x^{n,2})$. In this case, we would write $S^n = (\bar{\mu}^n, \bar{\sigma}^n)$ where $\bar{\mu}^n$ and $\bar{\sigma}^n$ are both vectors over all values of x.

Given our (belief) state S^n, we then have to define a policy (we might also call this a rule, or it might be a form of algorithm) that we denote by $X^\pi(S^n)$ where $x^n = X^\pi(S^n)$ is the decision we are going to use in our next trial where we either observe W^{n+1} or $\hat{F}^{n+1}$. While we would like to run this policy until $n \rightarrow \infty$, in practice we are going to be limited to N trials which then gives us a solution $x^{\pi,N}$. This solution depends on our initial state S^0, the observations $W^1, \ldots, W^N$ which occurred while we were finding $x^{\pi,N}$, and then we observe $\widehat{W}$ to evaluate $x^{\pi,N}$. We want to find the policy that solves

$$\max_\pi \mathbb{E}_{S^0}\mathbb{E}_{W^1,\ldots,W^N|S^0}\mathbb{E}_{\widehat{W}|S^0}\mathbb{E}_W F(x^{\pi,N}, \widehat{W}). \tag{2.41}$$

Basic newsvendor - cumulative reward A more realistic presentation of an actual newsvendor problem recognizes that we are accumulating profits while simultaneously learning about the demand W (or the function $\mathbb{E}_W F(x,W)$). If this is the case, then we would want to find a policy that solves

$$\max_\pi \mathbb{E}_{S_0}\mathbb{E}_{W_1,\ldots,W_T|S_0} \sum_{t=0}^{T-1} F(X^\pi(S_t), W_{t+1}). \tag{2.42}$$

Contextual newsvendor Imagine a newsvendor problem where the price p of our product is dynamic, given by p_t, which is revealed before we have to make a decision. Our profit would be given by

$$F(x, W|S_t) = p_t \min\{x, W\} - cx. \tag{2.43}$$

As before, assume that we do not know the distribution of W, and let B_t be the state of our belief about W (or about $\mathbb{E}F(x,W)$). Our state $S_t = (p_t, B_t)$, since we have to capture both the price p_t and our state of belief B_t. We can write our problem now as

$$\max_x \mathbb{E}_W F(x, W|p_t).$$

Now, instead of finding the optimal order quantity x^*, we have to find the optimal order quantity as a function of the price p_t, which we might write as $x^*(p_t)$. While x^* is a deterministic value, $x^*(p)$ is a function of price which represents the "context" for the decision x^*.

Multidimensional newsvendor problems Newsvendor problems can be multidimensional. One version is the *additive newsvendor problem* where there are K products to serve K demands, but using a production process that limits the total amount delivered. This would be formulated as

$$F(x_1, \ldots, x_K) = E_{W_1,\ldots,W_K} \sum_{k=1}^{K} p_k \min(x_k, W_k) - c_k x_k, \tag{2.44}$$

where

$$\sum_{k=1}^{K} x_k \leq U. \tag{2.45}$$

A second version arises when there are multiple products (different types/colors of cars) trying to satisfy the same demand W. This is given by

$$F(x_1,\ldots,x_K) = \mathbb{E}_W \left\{ \sum_{k=1}^{K} p_k \min \left[x_k, \left(W - \sum_{\ell=1}^{k-1} x_\ell \right)^+ \right] - \sum_{k=1}^{K} c_k x_k \right\}, \quad (2.46)$$

where $(Z)^+ = \max(0, Z)$.

2.3.2 Inventory/storage problems

Inventory (or storage) problems represent an astonishingly broad class of applications that span any problem where we buy/acquire (or sell) a resource to meet a demand, where excess inventory can be held to the next time period. Elementary inventory problems (with discrete quantities) appear to be the first problem to illustrate the power of a compact state space, which overcomes the exponential explosion that occurs if you try to formulate and solve these problems as decision trees.

Inventory without lags The simplest problem allows us to order new product x_t at time t that arrives right away.

R_t = Amount of inventory left over at the end of period t,
x_t = Amount ordered at the end of period t that will be available at the beginning of time period $t+1$,
$\hat{D}_{t+1}$ = Demand for the product that arises between t and $t+1$.
c_t = The unit cost of order product for product ordered at time t,
p_t = The price we are paid when we sell a unit during the period $(t, t+1)$.

Our basic inventory process is given by

$$R_{t+1} = \max\{0, R_t + x_t - \hat{D}_{t+1}\}.$$

We add up our total contribution at the end of each period. Let y_t be the sales during time period $(t-1, t)$. Our sales are limited by the demand $\hat{D}_t$ as well as our available product $R_{t-1} + x_{t-1}$, but we are going to allow ourselves to choose how much to sell, which may be smaller than either of these. So we would write

$$\begin{aligned} y_t &\leq R_{t-1} + x_{t-1}, \\ y_t &\leq \hat{D}_t. \end{aligned}$$

We are going to assume that we determine y_t at time t after we have learned the demands D_t for the preceding time period. So, at time t, the revenues and costs are given by

$$C_t(x_t, y_t) = p_t y_t - c_t x_t.$$

If this were a deterministic problem, we would formulate it as

$$\max_{(x_t,y_t),t=0,\ldots,T} \sum_{t=0}^{T} (p_t y_t - c_t x_t).$$

However, we often want to represent the demands $\hat{D}_{t+1}$ as being random at time t. We might want to allow our prices p_t, and perhaps even our costs c_t, to vary over time with both predictable (e.g. seasonal) and stochastic (uncertain) patterns. In this case, we are going to need to define a state variable S_t that captures what we know at time t before we make our decisions x_t and y_t. Designing state variables is subtle, but for now we would assume that it would include R_t, p_t, c_t, as well as the demands D_{t+1} that have arisen during interval $(t, t+1)$.

Unlike the newsvendor problem, the inventory problem can be challenging even if the distribution of demand D_t is known. However, if it is unknown, then we may need to maintain a belief state B_t about the distribution of demand, or perhaps the expected profits when we place an order x_t.

The features of this problem allow us to create a family of problems:

Static data If the prices p_t and costs c_t are constant (which is to say that $p_t = p$ and $c_t = c$), with a known distribution of demand, then we have a stochastic optimization problem where the state is just $S_t = R_t$.

Dynamic data Assume the price p_t evolves randomly over time, where $p_{t+1} = p_t + \varepsilon_{t+1}$, then our state variable is $S_t = (R_t, p_t)$.

History-dependent processes Imagine now that our price process evolves according to

$$p_{t+1} = \theta_0 p_t + \theta_1 p_{t-1} + \theta_2 p_{t-2} + \varepsilon_{t+1},$$

then we would write the sate as $S_t = \big(R_t, (p_t, p_{t-1}, p_{t-2})\big)$.

Learning process Now assume that we do not know the distribution of the demand. We might put in place a process to try to learn it, either from observations of demands or sales. Let B_t capture our belief about the distribution of demand, which may itself be a probability distribution. In this case, our state variable would be $S_t = (R_t, p_t, B_t)$.

Let $Y^\pi(S_t)$ be the selling policy we use to determine y_t, and let $X^\pi(S_t)$ be the buying policy we use for determining x_t, where π carries the parameters that determine both policies. We would write our objective function as

$$\max_\pi \mathbb{E} \sum_{t=0}^{T} (p_t Y^\pi(S_t) - c_t X^\pi(S_t)).$$

Inventory problems are quite rich. This is a problem where it is quite easy to create variations that can be solved with each of the four classes of policies introduced in section 1.4. We describe these four classes of policies in much more depth in chapter 11. In section 11.9, we illustrate an inventory problem that arises in energy storage where each of the four classes of policies may work best.

Inventory planning with forecasts An important extension that arises in many real applications is where the data (demands, prices, even costs) may follow time-varying patterns which can be approximately forecasted. Let

$$f^W_{tt'} = \text{Forecast of some activity (demands, prices, costs) made at time } t \text{ that we think will happen at time } t'.$$

Forecasts evolve over time. They may be given to us from an exogenous source (a forecasting vendor), or we may use observed data to do our own updating of forecasts.

Assuming they are provided by an external vendor, we might describe the evolution of forecasts using

$$f^W_{t+1,t'} = f^W_{tt'} + \hat{f}^W_{t+1,t'},$$

where $\hat{f}^W_{t+1,t'}$ is the (random) change in the forecasts over all future time periods t'.

When we have forecasts, the vector $f^W_t = (f^W_{tt'})_{t' \geq t}$ technically becomes part of the state variable. When forecasts are available, the standard approach is to treat these as latent variables, which means that we do not explicitly model the evolution of the forecasts, but rather just treat the forecast as a static vector. We will return to this in chapter 9, and describe a strategy for handling rolling forecasts in chapter 13.

Lagged decisions There are many applications where we make a decision at time t (say, ordering new inventory) that does not arrive until time t' (as a result of shipping delays). In global logistics, these lags can extend for several months. For an airline ordering new aircraft, the lags can span several years.

We can represent lags using the notation

$x_{tt'}$ = Inventory ordered at time t to arrive at time t'.

$R_{tt'}$ = Inventory that has been ordered at some time before t that is going to arrive at time t'.

The variable $R_{tt'}$ is how we capture the effect of previous decisions. We can roll these variables up into the vectors $x_t = (x_{tt'})_{t' \geq t}$ and $R_t = (R_{tt'})_{t' \geq t}$.

Lagged problems are particularly difficult to model. Imagine that we want to sign contracts to purchase natural gas in month t'' that might be three years into the future to serve uncertain demands. This decision has to consider the possibility that we may place an order $x_{t't''}$ at a time t' that is between now (time t) and time t''. At time t, the decision $x_{t't''}$ is a random variable that depends not just on the price of natural gas at time t', but also the decisions we might make between t and t', as well as evolving forecasts.

2.3.3 Shortest path problems

Shortest path problems represent a particularly elegant and powerful problem class, since a node in the network can represent any discrete state, while links out of the node can represent a discrete action.

A deterministic shortest path problem A classical sequential decision problem is the shortest path problem. Let

$\mathcal{I}$ = The set of nodes (intersections) in the network,

$\mathcal{L}$ = The set of links (i, j) in the network,

c_{ij} = The cost (typically the time) to drive from node i to node j, $i, j \in \mathcal{I}, (i, j) \in \mathcal{L}$,

$\mathcal{I}^+_i$ = The set of nodes j for which there is a link $(i, j) \in \mathcal{L}$,

$\mathcal{I}^-_j$ = The set of nodes i for which there is a link $(i, j) \in \mathcal{L}$.

A traveler at node i needs to choose the link (i, j) where $j \in \mathcal{I}^+_i$ is a downstream node from node i. Assume that the traveler needs to get from an origin node q to a destination node r at least cost. Let

v_j = The minimum cost required to get from node j to node r.

Step 0. Let

$$v_j^0 = \begin{cases} \text{M} & j \neq r, \\ 0 & j = r. \end{cases}$$

where "M" is known as "big-M" and represents a large number. Let $n = 1$.

Step 1. Solve for all $i \in \mathcal{I}$,

$$v_i^n = \min_{j \in \mathcal{I}_i^+} \left(c_{ij} + v_j^{n-1} \right).$$

Step 2. If $v_i^n < v_i^{n-1}$ for any i, let $n = n + 1$ and return to step 1. Else stop.

Figure 2.4 A basic shortest path algorithm.

We can think of v_j as the value of being in state j. At optimality, these values will satisfy

$$v_i = \min_{j \in \mathcal{I}_i^+} \left(c_{ij} + v_j \right).$$

This fundamental equation underlies all the shortest path algorithms used in navigation systems, although these have been heavily engineered to achieve the rapid response we have become accustomed to. A basic shortest path algorithm is given in 2.4, although this represents just the skeleton of what a real algorithm would look like.

A stochastic shortest path problem We are often interested in shortest path problems where there is uncertainty in the cost of traversing a link. For our transportation example, it is natural to view the travel time on a link as random, reflecting the variability in traffic conditions on each link.

To handle this new dimension correctly, we have to specify whether we see the outcome of the random cost on a link before or after we make the decision whether to traverse the link. If the actual cost is only realized after we traverse the link, then our decision at node x_i that we made when we are at node i would be written

$$x_i = \arg \min_{j \in \mathcal{I}_i^+} \mathbb{E} \left(\hat{c}_{ij} + v_j \right),$$

where the expectation is over the (assumed known) distribution of the random cost $\hat{c}_{ij}$. For this problem, our state variable S is simply the node at which we are located.

If we get to make our decision after we learn $\hat{c}_{ij}$, then our decision would be written

$$x_i = \mathbb{E} \left\{ \arg \min_{j \in \mathcal{I}_i^+} \left(\hat{c}_{ij} + v_j \right) \right\}.$$

In this setting, the state variable S is given by $S = (i, (\hat{c}_{ij})_j)$ includes both our current node, but also the costs on links emanating from node i.

A dynamic shortest path problem Now imagine the problem being solved by any online navigation system which gets live information from the network, and updates the shortest path periodically. Assume at time t that the navigation system has estimates $\bar{c}_{tij}$ of the cost of traversing link $(i, j) \in \mathcal{L}$ where $\mathcal{L}$ is the set of all the links in the network.

The system uses these estimates to solve a deterministic shortest path problem which recommends what to do right now.

Assume that the vector of estimated costs $\bar{c}_t$ is updated each time period (perhaps this is every 5 minutes), so at time $t+1$ we are given the vector of estimates $\bar{c}_{t+1}$. Let N_t be the node where the traveler is located (or is heading inbound to). The state variable is now

$$S_t = (N_t, \bar{c}_t).$$

Remembering that there is an element of $\bar{c}_t$ for each link in the network, our state variable S_t has dimensionality $|\mathcal{L}| + 1$. In chapter 19 we will describe how it is that we can solve such a complex problem using simple shortest path calculations.

A robust shortest path problem We know that costs c_{ij} are uncertain. The navigation services can use their observations to build probability distributions for $\bar{c}_{tij}$ for the estimates of the travel times given what we know at time t. Now, imagine that, rather than taking an average, we use the θ-percentile, which we represent by $\bar{c}_{tij}(\theta)$. So, if we set $\theta = 0.90$, we would be using the $90th$ percentile travel time, which would discourage using links that can become highly congested.

Now let $\ell_t^\pi(\theta) \in \mathcal{L}$ be the link that is recommended when we are in state $S_t = (N_t, \bar{c}_t(\theta))$ and choose a direction by solving a deterministic shortest path problem using the link costs $\bar{c}_t(\theta)$. Let $\hat{c}_{t,\ell_t^\pi(\theta)}$ be the actual cost the traveler experiences traversing link $\ell_t^\pi(\theta) = (i,j) \in \mathcal{L}$ at time t. The problem is now to optimize across this class of policies by solving

$$\min_\theta \mathbb{E}\left\{\sum_t \hat{c}_{t,\ell_t^\pi(\theta)} | S_0\right\},$$

where S_0 captures the starting point of the vehicle and initial estimates of the costs. We discuss this strategy in further depth in chapter 19.

2.3.4 Some fleet management problems

Fleet management problems, such as those that arise with ride hailing fleets, represent a special class of resource allocation problem. In this section we start by describing the problem faced by a single truck driver we call the "nomadic trucker," and then show how to extend the basic idea to fleets of trucks.

The nomadic trucker The nomadic trucker is a problem where a single truck driver will pick up a load at A, drive it from A to B, drop it off at B, and then has to look for a new load (there are places to call in to get a list of available loads). The driver has to think about how much money he will make moving the load, but he then also has to recognize that the load will move him to a new city.

The driver is characterized at each point in time by his current or future location ℓ_t (which is a region of the country), his equipment type E_t which is the type of trailer he is pulling (which can change depending on the needs of the freight), his estimated time of arrival at ℓ_t (denoted by τ_t^{eta}), and the time τ_t^{home} that he has been away from his home (which is fixed, so we exclude it from the state variable). We roll these attributes into an attribute vector a_t given by

$$a_t = (\ell_t, E_t, \tau_t^{eta}, \tau_t^{home}).$$

When the driver arrives at the destination of a load, he calls a freight broker and gets a set of loads $\mathcal{L}_t$ that he can choose from. This means that his state variable (the information just before he makes a decision), is given by

$$S_t = (a_t, \mathcal{L}_t).$$

The driver has to choose among a set of actions $\mathcal{X}_t = (\mathcal{L}_t, \text{"hold"})$ that includes the loads in the set $\mathcal{L}_t$, or doing nothing. Once the driver makes this choice, the set $\mathcal{L}_t$ is no longer relevant. His state immediately after he makes his decision is called the *post-decision state* $S_t^x = a_t^x$, which is updated to reflect the destination of the load, and the time he is expected to arrive at this location.

The natural way for a driver to choose which action to take is to balance the contribution of the action, which we write as $C(S_t, x_t)$, and the value of the driver in his "post-decision" state a_t^x. We might write this policy, which we call $X^\pi(S_t)$, using

$$X^\pi(S_t) = \arg\max_{x \in \mathcal{X}_t} \big(C(S_t, x) + \overline{V}_t^x(a_t^x)\big). \tag{2.47}$$

The algorithmic challenge is creating the estimates $\overline{V}_t^x(a_t^x)$, which is an example of what we will call a *value function approximation*. If the number of possible values of the driver attribute vector a_t^x was not too large, we could solve this problem using the same way we would solve the stochastic shortest path problem introduced in section 2.3.3. The hidden assumption in this problem is that the number of nodes is not too large (even a million nodes is considered manageable). When a "node" is a multidimensional vector a_t, then we may have trouble manipulating all the possible values this may take (another instances of the curse of dimensionality).

From one driver to a fleet We can model a fleet of drivers by defining

$$\begin{aligned} R_{ta} &= \text{The number of drivers with attribute vector } a \text{ at time } t, \\ R_t &= (R_{ta})_{a \in \mathcal{A}}. \end{aligned}$$

where $a \in \mathcal{A}$ is in an attribute space that spans all the possible values that each element of a_t may take.

Similarly, we might describe loads by an attribute vector b that contains information such as origin, destination, scheduled pickup and delivery windows, required equipment type, and whether the load contains hazardous materials. In the United States, it is typical to aggregate the country into 100 regions, giving us 10,000 origin-destination pairs. Let

$$\begin{aligned} L_{tb} &= \text{The number of loads with attribute vector } b \text{ at time } t, \\ L_t &= (L_{tb})_{b \in \mathcal{B}}. \end{aligned}$$

Our state variable is then given by

$$S_t = (R_t, L_t).$$

We leave it as an exercise to the reader to try to estimate the size of the state space for this problem. We show how this problem can be solved in chapter 18 using value function approximations.

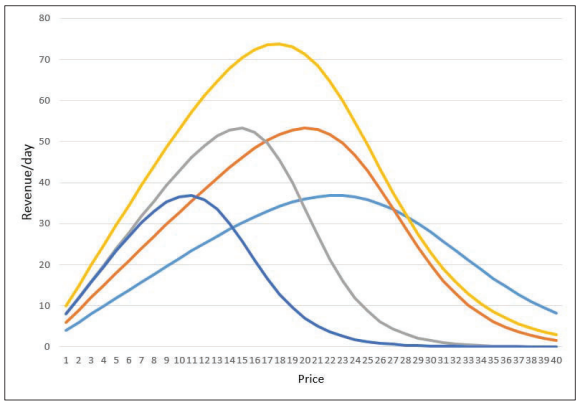

Figure 2.5 Illustration of a family of possible revenue curves.

2.3.5 Pricing

Imagine that we are trying to determine the price of a product, and that we feel that we can model the demand for the product using a logistics curve given by

$$D(p|\theta) = \theta_0 \frac{e^{\theta_1 - \theta_2 p}}{1 + e^{\theta_1 - \theta_2 p}}.$$

The total revenue from charging price p is given by

$$R(p|\theta) = pD(p|\theta).$$

If we knew θ, finding the optimal price would be a fairly simple exercise. But now assume that we do not know θ. Figure 2.5 illustrates a family of potential curves that might describe revenue as a function of price.

We can approach this problem as one of learning the true value of θ. Let $\Theta = (\theta_1, \ldots, \theta_K)$ be a family of possible values of θ where we assume that one of the elements of Θ is the true value. Let p_k^n be the probability that $\theta = \theta_k$ after we have made n observations. The state of our learning system, then, is $S^n = (p_k^n)_{k=1}^K$ which captures our belief about θ. We revisit this problem in chapter 7.

2.3.6 Medical decision making

Physicians have to make decisions about patients who arrive with some sort of complaint. The process starts by taking a medical history which consists of a series of questions about the patients history and lifestyle. Let h^n be this history, where h^n might consist of thousands of different possible characteristics (humans are complicated!). The physician might then order additional tests which produce additional information, or she might prescribe medication or request a surgical procedure. Let d^n capture these decisions. We can wrap this combination of patient history h^n and medical decisions d^n into a set of explanatory variables that we designate $x^n = (h^n, d^n)$. Also let θ be a parameter vector with the same dimensionality as x^n.

Now assume we observe an outcome y^n which for simplicity we are going to represent as binary, where $y^n = 1$ can be interpreted as "success" and $y^n = 0$ is a "failure." We

are going to assume that we can model the random variable y^n (random, that is, before we observe the results of the treatment) using a logistic regression model, which is given by

$$\mathbb{P}[y^n = 1|x^n = (h^n, d^n), \theta] = \frac{e^{\theta^T x^n}}{1 + e^{\theta^T x^n}}. \tag{2.48}$$

This problem illustrates two types of uncertainty. The first is the patient history h^n, where we typically would not have a probability distribution describing these attributes. It is difficult (actually, impossible) to develop a probabilistic model of the complex characteristics captured in h^n describing a person, since a history is going to exhibit complex correlations. By contrast, the random variable y^n has a well defined mathematical model, characterized by an unknown (and high dimensional) parameter vector θ.

We can use two different approaches for handling these different types of uncertainty. For patient attributes, we are going to use an approach that is often known as *data driven*. We might have access to a large dataset of prior attributes, decisions and outcomes, that we might represent as $(x^n = (h^n, d^n), y^n)_{n=1}^N$. Alternatively, we may assume that we simply observe a patient h^n (this is the data-driven part), then make a decision $d^n = D^\pi(S^n)$ using a decision function $D^\pi(S^n)$ that can depend on a state variable S^n, and then observe an outcome y^n which we can describe using our probability model.

2.3.7 Scientific exploration

Scientists looking to discover new drugs, new materials, or new designs for a wing or rocket engine, are often faced with the need to run difficult laboratory experiments looking for the inputs and processes to produce the best results. Inputs might be a choice of catalyst, the shape of a nanoparticle, or the choice of molecular compound. There might be different steps in a manufacturing process, or the choice of a machine for polishing a lens.

Then, there are the continuous decisions. Temperatures, pressures, concentrations, ratios, locations, diameters, lengths and times are all examples of continuous parameters. In some settings these are naturally discretized, although this can be problematic if there are three or more continuous parameters we are trying to tune at the same time.

We can represent a discrete decision as choosing an element $x \in \mathcal{X} = \{x_1, \ldots, x_M\}$. Alternatively, we may have a continuous vector $x = (x_1, x_2, \ldots, x_K)$. Let x^n be our choice of x (whether it is discrete or continuous). We are going to assume that x^n is the choice we make *after* running the n^{th} experiment that guides the $n + 1^{st}$ experiment, from which we observe W^{n+1}. The outcome W^{n+1} might be the strength of a material, the reflexivity of a surface, or the number of cancer cells killed.

We use the results of an experiment to update a belief model. If x is discrete, imagine we have an estimate $\bar{\mu}_x^n$ which is our estimate of the performance of running an experiment with choice x. If we choose $x = x^n$ and observe W^{n+1}, then we can use statistical methods (which we describe in chapter 3) to obtain updated estimates $\bar{\mu}_x^{n+1}$. In fact, we can use a property known as *correlated beliefs* that may allow us to run experiment $x = x^n$ and update estimates $\bar{\mu}_{x'}^{n+1}$ for values x' other than x.

Often, we are going to use some parametric model to predict a response. For example, we might create a linear model which can be written

$$f(x^n|\theta) = \theta_0 + \theta_1\phi_1(x^n) + \theta_2\phi_2(x^n) + \ldots, \tag{2.49}$$

where $\phi_f(x^n)$ is a function that pulls out relevant pieces of information from the inputs x^n of an experiment. For example, if element x_i is the temperature, we might have $\phi_1(x^n) = x_i^n$

and $\phi_2(x^n) = (x_i^n)^2$. If x_{i+1} is the pressure, we could also have $\phi_3(x^n) = x_i^n x_{i+1}^n$ and $\phi_4(x^n) = x_i^n(x_{i+1}^n)^2$.

Equation (2.49) is known as a linear model because it is linear in the parameter vector θ. The logistic regression model in (2.48) is an example of a nonlinear model (since it is nonlinear in θ). Whether it is linear or nonlinear, parametric belief models capture the structure of a problem, reducing the uncertainty from an unknown $\bar{\mu}_x$ for each x (where the number of different values of x can number in the thousands to millions or more) down to a set of parameters θ that might number in the tens to hundreds.

2.3.8 Machine learning vs. sequential decision problems

There are close parallels between designing policies for sequential decision problems and machine learning. Let:

x^n = the data corresponding to the n^{th} instances of a problem (the characteristics of a patient, the attributes of a document, the data for an image) that we want to use to predict an outcome y^n,

y^n = the response, which might be the response of a patient to a treatment, the categorization of a document, or the classification of an image,

$f(x^n|\theta)$ = our model which we use to predict y^n given x^n,

θ = an unknown parameter vector used to determine the model.

We assume we have some metric that indicates how well our model $f(x|\theta)$ is performing. For example, we might use

$$L(x^n, y^n|\theta) = (y^n - f(x^n|\theta))^2.$$

The function $f(x|\theta)$ can take on many forms. The simplest is a basic linear model of the form

$$f(x|\theta) = \sum_{f\in\mathcal{F}} \theta_f \phi_f(x),$$

where $\phi_f(x)$ is known as a feature, and $\mathcal{F}$ is the set of features. There may be just a handful of features, or thousands. The statistics and machine learning communities have developed a broad array of functions, each of which is parameterized by some vector θ (sometimes designated as weights w). We review these in some depth in chapter 3.

The machine learning problem is to first pick a class of statistical model $f \in \mathcal{F}$, and then tune the parameters $\theta \in \Theta^f$ associated with that class of function. We write this as

$$\min_{f\in\mathcal{F},\theta\in\Theta^f} \frac{1}{N}\sum_{n=1}^{N}(y^n - f(x^n|\theta))^2. \tag{2.50}$$

When we are solving a sequential decision problem, we need to find the best policy. We can think of a policy π as consisting of choosing a function $f \in \mathcal{F}$ along with tunable parameters $\theta \in \Theta^f$. When we write our problem of optimizing over policies, we typically use

$$\max_{\pi=(f\in\mathcal{F},\theta\in\Theta^f)} \mathbb{E}\left\{\sum_{t=0}^{T} C(S_t, X^\pi(S_t|\theta))|S_0\right\}. \tag{2.51}$$

When we compare the machine learning problem (2.50) with the sequential decision problem (2.51), we see that both are searching over classes of functions. We argue in chapter 3 that there are three (overlapping) classes of functions used for machine learning: lookup tables, parametric and nonparametric functions. Then we are going to argue in chapter 11 that there are four classes of policies (that is, four set of functions in $\mathcal{F}$ when we are designing policies), where one of them, policy function approximations, includes all the functions that we might use in machine learning. The other three are all forms of optimization problems.

2.4 BIBLIOGRAPHIC NOTES

Section 2.1.1 - The field of stochastic search traces its roots to two papers: Robbins & Monro (1951) for derivative-based stochastic search, and Box & Wilson (1951) for derivative-free methods. Some early papers include the work on unconstrained stochastic search including Wolfowitz (1952) (using numerical derivatives), Blum (1954*a*) (extending to multidimensional problems), and Dvoretzky (1956), which contributed theoretical research. A separate line of research focused on constrained problems under the umbrella of "stochastic quasi-gradient" methods, with seminal contributions from Ermoliev (1968), Shor (1979), Pflug (1988), Kushner & Clark (1978), Shapiro & Wardi (1996), and Kushner & Yin (2003). As with other fields, this field broadened over the years. The best modern review of the field (under this name) is Spall (2003*a*), which was the first book to pull together the field of stochastic search as it was understood at that time. Bartlett et al. (2007) approaches this topic from the perspective of online algorithms, which refers to stochastic gradient methods where samples are provided by an exogenous source.

The derivative-free version of stochastic search with discrete alternatives has been widely studied as the *ranking and selection* problem. Ranking and selection enjoys a long history dating back to the 1950's, with an excellent treatment of this early research given by the classic DeGroot (1970), with a more up to date review in Kim & Nelson (2007). Recent research has focused on parallel computing (Luo et al. (2015), Ni et al. (2016)) and handling unknown correlation structures (Qu et al., 2012). However, ranking and selection is just another name for derivative-free stochastic search, and has been widely studied under this umbrella (Spall, 2003*a*). The field has attracted considerable attention from the simulation-optimization community, reviewed next.

Section 2.1.2 Decision trees represent the simplest approach to modeling and, for simple settings, solving sequential decision problems. They lend themselves to complex decision problems in health (should a patient receive an MRI?), business (should a business enter a new market) and policy (should the military pursue a new strategy). (Skinner, 1999) is one of many books on decision trees, and there are literally dozens of survey articles addressing the use of decision trees in different application areas.

Section 2.1.3 - The field of Markov decision processes was introduced, initially in the form of deterministic dynamic programs, by Bellman (1952), leading to his classic reference (Bellman, 1957) (see also (Bellman, 1954) and (Bellman et al., 1955)), but this work was continued by a long stream of books including Howard (1960) (another classic), Nemhauser (1966), Denardo (1982), Heyman & Sobel (1984), leading up to Puterman (2005*a*) (this first appeared in 1994). Puterman's book represents the last

but best in a long series of books on Markov decision processes, and now represents the major reference in what is a largely theoretical field, since the core of the field depends on one-step transition matrices which are rarely computable, and only for extremely small problems. More recently, Bertsekas (2017) provides an in-depth summary of the field of dynamic programming and Markov decision processes using a style that is a hybrid of notation from optimal control, with the principles of Markov decision processes, while also covering many of the concepts from approximate dynamic programming and reinforcement learning (covered below).

Section 2.1.4 - There is a long history in the development of optimal control dating to the 1950s, summarized by many books including Kirk (2012), Stengel (1986), Sontag (1998), Sethi (2019), and Lewis et al. (2012). The canonical control problem is continuous, low-dimensional and unconstrained, which leads to an analytical solution. Of course, applications evolved past this canonical problem, leading to the use of numerical methods. Deterministic optimal control is widely used in engineering, whereas stochastic optimal control has tended to involve much more sophisticated mathematics. Some of the most prominent books include Astrom (1970), Kushner (1971), Bertsekas & Shreve (1978), Yong & Zhou (1999), Nisio (2014) and Bertsekas (2017) (note that some of the books on deterministic controls touch on the stochastic case).

As a general problem, stochastic control covers any sequential decision problem, so the separation between stochastic control and other forms of sequential stochastic optimization tends to be more one of vocabulary and notation (Bertsekas (2017) is a good example of a book that bridges these vocabularies). Control-theoretic thinking has been widely adopted in inventory theory and supply chain management (e.g. Ivanov & Sokolov (2013) and Protopappa-Sieke & Seifert (2010)), finance (Yu et al., 2010), and health services (Ramirez-Nafarrate et al., 2014), to name a few.

There is considerable overlap between the fields of dynamic programming (including Markov decision processes) and optimal control (including stochastic control), but the two fields have evolved largely independently, using different notation, and motivated by very different applications. However, there are numerous parallels in the development of numerical methods for solving problems in both fields. Both fields start from the same foundation, known as Bellman's equations in dynamic programming, and Hamilton-Jacobi equations in optimal control (leading some to refer to them as Hamilton-Jacobi-Bellman (or HJB) equations.

Section 2.1.5 - Approximate dynamic programming (also referred to as adaptive dynamic programming and, for a period, neuro-dynamic programming) has been studied since Bellman first recognized that discrete dynamic programming suffered from the curse of dimensionality (see Bellman & Dreyfus (1959) and Bellman et al. (1963)), but the operations research community then seemed to drop any further research in approximation methods until the 1980's. As computers improved, researchers began tackling Bellman's equation using numerical approximation methods, with the most comprehensive presentation in Judd (1998) which summarized almost a decade of research (see also Chen et al. (1999)).

A completely separate line of research in approximations evolved in the control theory community with the work of Paul Werbos (Werbos (1974)) who recognized that the "cost-to-go function" (the same as the value function in dynamic programming), could be approximated using various techniques. Werbos helped develop this area

through a series of papers (examples include Werbos (1989), Werbos (1990), Werbos (1992*a*) and Werbos (1994)). Important references are the edited volumes (White & Sofge, 1992) and (Si et al., 2004*a*) which highlighted what had already become a popular approach using neural networks to approximate both policies ("actor nets") and value functions ("critic nets"). Si et al. (2004*b*) contains a nice review of the field as of 2002.

Tsitsiklis (1994) and Jaakkola et al. (1994*a*) were the first to recognize that the basic algorithms being developed under the umbrella of reinforcement learning represented generalizations of the early stochastic gradient algorithms of Robbins & Monro (1951). Bertsekas & Tsitsiklis (1996) laid the foundation for adaptive learning algorithms in dynamic programming, using the name "neuro-dynamic programming." Werbos, (e.g. Werbos (1992*a*)), had been using the term "approximate dynamic programming," which became the title of Powell (2007) (with a major update in Powell (2011)), a book that also merged math programming and value function approximations to solve high-dimensional, convex stochastic optimization problems (but, see the developments under stochastic programming below). Later, the engineering controls community reverted to "adaptive dynamic programming" as the operations research community adopted "approximate dynamic programming."

Section 2.1.6 - A third line of research into approximation methods started in the 1980s in the computer science community under the umbrella of "reinforcement learning" with the work of Richard Sutton and Andy Barto into Q-learning. The field took off with the appearance of their now widely cited book (Sutton & Barto, 2018), although by this time the field was quite active (see the review Kaelbling et al. (1996)). Research under the umbrella of "reinforcement learning" has evolved to include other algorithmic strategies under names such as policy search and Monte Carlo tree search. Other references from the reinforcement learning community include Busoniu et al. (2010) and Szepesvári (2010*a*). In 2017, Bertsekas published the fourth edition of his optimal control book (Bertsekas (2017)), which covers a range of topics spanning classical Markov decision processes and the approximate algorithms associated with approximate dynamic programming and optimal control, but using the notation of optimal control and constructs from Markov decision processes (such as one-step transition matrices). Bertsekas' book easily has the most comprehensive review of the ADP/RL literature, and we recommend that readers looking for a comprehensive bibliography of these fields (as of 2017). In 2018, Sutton and Barto came out with a greatly expanded second edition of their classic *Reinforcement Learning* book (Sutton & Barto (2018)) which features methods that move far behind the basic Q-learning algorithms of the first edition. In the language of this book, readers comparing the first and second editions of *Reinforcement Learning* will see the transition from policies based on value functions alone (Q-learning in the RL community), to examples from all four classes of policies.

Section 2.1.7 - Optimal stopping is an old and classic topic. An elegant presentation is given in Cinlar (1975) with a more recent discussion in Cinlar (2011) where it is used to illustrate filtrations. DeGroot (1970) provides a nice summary of the early literature. One of the earliest books dedicated to the topic is Shiryaev (1978) (originally in Russian). Moustakides (1986) describes an application to identifying when a stochastic process has changed, such as the increase of incidence in a disease or a drop in quality on a production line. Feng & Gallego (1995) uses optimal

stopping to determine when to start end-of-season sales on seasonal items. There are numerous uses of optimal stopping in finance (Azevedo & Paxson, 2014), energy (Boomsma et al., 2012) and technology adoption (Hagspiel et al., 2015), to name just a few.

Section 2.1.8 - There is an extensive literature exploiting the natural convexity of $Q(x_0, W_1)$ in x_0, starting with Van Slyke & Wets (1969), followed by the seminal papers on stochastic decomposition (Higle & Sen, 1991) and the stochastic dual dynamic programming (SDDP) (Pereira & Pinto, 1991). A substantial literature has unfolded around this work, including Shapiro (2011) who provides a careful analysis of SDDP, and its extension to handle risk measures (Shapiro et al. (2013), Philpott et al. (2013)). A number of papers have been written on convergence proofs for Benders-based solution methods, but the best is Girardeau et al. (2014). Kall & Wallace (2009) and Birge & Louveaux (2011) are excellent introductions to the field of stochastic programming. King & Wallace (2012) is a nice presentation on the process of modeling problems as stochastic programs. A modern overview of the field is given by Shapiro et al. (2014*a*). ,

Section 2.1.9 - Active learning problems have been studied as "multiarmed bandit problems" since 1960 in the applied probability community. DeGroot (1970) was the first to show that an optimal policy for the multiarmed bandit problem could be formulated (if not solved) using Bellman's equation (this is true of *any* learning problem, regardless of whether we are maximizing final or cumulative rewards). The first real breakthrough occurred in Gittins & Jones (1974) (the first and most famous paper), followed by Gittins (1979). The theory of Gittins indices was described thoroughly in his first book (Gittins, 1989), but the "second edition" (Gittins et al., 2011), which was a complete rewrite of the first edition, represents the best introduction to the field of Gittins indices, which now features hundreds of papers. However, the field is mathematically demanding, with index policies that are difficult to compute.

A parallel line of research started in the computer science community with the work of Lai & Robbins (1985*a*) who showed that a simple policy known as *upper confidence bounding* possessed the property that the number of times we test the wrong arm can be bounded (although it continues to grow with n). The ease of computation, combined with these theoretical properties, made this line of research extremely attractive, and has produced an explosion of research. While no books on this topic have appeared as yet, an excellent monograph is Bubeck & Cesa-Bianchi (2012).

These same ideas have been applied to bandit problems using a terminal reward objective using the label the "best arm" bandit problem (see Audibert & Bubeck (2010), Kaufmann et al. (2016), Gabillon et al. (2012)).

Section 2.1.10 - The original work on optimal computing budget allocation was developed by Chun-Hung Chen in Chen (1995), followed by a series of articles (Chen (1996), Chen et al. (1997), Chen et al. (1998), Chen et al. (2003), Chen et al. (2008)), leading up to the book Chen & Lee (2011) that provides a thorough overview of this field. The field has focused primarily on discrete alternatives (e.g. different designs of a manufacturing system), but has also included work on continuous alternatives (e.g. Hong & Nelson (2006)). An important recent result by Ryzhov (2016) shows the asymptotic equivalence of OCBA and expected improvement policies which maximize the value of information. When the number of alternatives is much larger

(say, 10,000), techniques such as simulated annealing, genetic algorithms and tabu search (adapted for stochastic environments) have been brought to bear. Swisher et al. (2000) contains a nice review of this literature. Other reviews include Andradóttir (1998*a*), Andradóttir (1998*b*), Azadivar (1999), Fu (2002), and Kim & Nelson (2007). The recent review Chau et al. (2014) focuses on gradient-based methods.

The scope of problems and methods studied under the umbrella of "simulation-optimization" has steadily grown (a pattern similar to other communities in stochastic optimization). The best evidence of this is Michael Fu's *Handbook of Simulation Optimization* (Fu (2014)) which is a superb reference for many of the tools in this field.

Section 2.1.11 - Active learning is a field that emerged from within the machine learning community; parallels the bandit community in that an agent could control (or influence) the inputs x^n to a learning process that produces observations y^n. The field emerged primarily in the 1990s (see in particular Cohn et al. (1996) and Cohn et al. (1994)). The book Settles (2010) provides a nice introduction to the field which indicates a strong awareness of the parallels between active learning and multiarmed bandit problems. A recent tutorial is given by Krempl et al. (2016).

Section 2.1.12 - Chance constrained optimization was first introduced by Charnes et al. (1959), followed by Charnes & Cooper (1963), for handling constraints that involve uncertainty. It has also been studied as "probabilistic constrained programming" (Prekopa (1971),Prekopa (2010)) and continues to attract hundreds of papers each year. Chance-constrained programming is standard in many books on stochastic optimization (see, for example, Shapiro et al. (2014*a*)).

Section 2.1.13 - This is a subfield of optimal control, but it evolved into a field of its own, with popular books such as Camacho & Bordons (2003) and thousands of articles (see Lee (2011) for a 30-year review). As of this writing, there are over 50 review articles feature modeling predictive control since 2010.

Section 2.1.14 - A thorough review of the field of robust optimization is contained in Ben-Tal et al. (2009) and Bertsimas et al. (2011), with a more recent review given in Gabrel et al. (2014). Bertsimas & Sim (2004) studies the price of robustness and describes a number of important properties. Robust optimization is attracting interest in a variety of application areas including supply chain management (Bertsimas & Thiele (2006), Keyvanshokooh et al. (2016)), energy (Zugno & Conejo, 2015). and finance (Fliege & Werner, 2014).

EXERCISES

Review questions

2.1 What is meant by the *compact form* and *expanded form* of the expectation operator? Give an illustration of each.

2.2 Write out the objective functions that we would use when maximizing the cumulative reward or maximizing the final reward.

2.3 Compare the Markov decision process model in section 2.1.3 to the optimal control model in section 2.1.4 by creating a table showing how each approach models the following:

- State variables,
- decision/control variables,
- the transition function (use the version in the optimal control formulation that includes the randomness w_t),
- the value of being in a state at time t,
- how this value can be used to find the best decision given the state x_t (otherwise known as the policy).

2.4 From the very brief presentation in this chapter, what is the difference between approximate dynamic programming, and reinforcement learning (using Q-learning).

2.5 Write out an optimal stopping problem as an optimal control problem. Would the optimal policy take the form in equation (2.12)? Justify your answer.

2.6 Does solving the optimization problem in (2.21) produce an optimal policy? Discuss why or why not.

2.7 In the stochastic programming model in section 2.22, what is meant by "ω"? Use the setting of allocating inventories to warehouses at time 0 (this decision is given by x_0), after which we see demands, and then determine which warehouse should satisfy each demand.

2.8 For the multiarmed bandit problem, write out the objective function for finding the best interval estimation policy.

2.9 Describe in words the decision that is being optimized over using the OCBA algorithm in simulation optimization. Contrast how OCBA operates (in general terms) compared to interval estimation for the multiarmed bandit problem.

2.10 What objective is being optimized in active learning? Could you solve this same problem using interval estimation?

2.11 What is the core computational challenge that arises in chance constrained programming?

2.12 Compare model predictive control to using stochastic programming as a policy.

2.13 Describe in words, using an example, the core idea of robust optimization. Just as the two stage stochastic program in (2.22) could be written as a policy (as we do in equation (2.24)) show how robust optimization can also be written as a policy.

2.14 From section 2.3.8, what is the difference between a machine learning problem, and a sequential decision problem?

Modeling questions

2.15 Provide three examples of:

a) Problems where we would want to maximize the cumulative reward (or minimize cumulative cost).

b) Problems where we would want to maximize the final reward (or minimize the final cost).

2.16 Show how to write solve a decision tree (section 2.1.2) as a Markov decision process (section 2.1.3) using Bellman's equation (2.6).

2.17 Put the contextual newsvendor problem in section 2.3.1 into the format of the universal modeling framework in section 2.2. Introduce and define any additional notation you may need.

2.18 Put the inventory planning problem with forecasts in section 2.3.2 into the format of the universal modeling framework in section 2.2. Introduce and define any additional notation you may need.

2.19 Put the dynamic shortest path problem in section 2.3.3 into the format of the universal modeling framework in section 2.2. Introduce and define any additional notation you may need.

2.20 Put the robust shortest path problem in section 2.3.3 into the format of the universal modeling framework in section 2.2. Introduce and define any additional notation you may need.

2.21 Put the nomadic trucker problem in section 2.3.4 into the format of the universal modeling framework in section 2.2. The state variable $S_t = (a_t, \mathcal{L}_t)$ given in the section is incomplete. What is missing? Introduce and define any additional notation you may need. [Hint: Carefully review the definition of the state variable given in section 2.2. Now look at the policy in (2.47), and see if there is any statistic that will be changing over time that is needed to make a decision (which means it has to go into the state variable).]

2.22 Put the pricing problem in section 2.3.5 into the format of the universal modeling framework in section 2.2. Introduce and define any additional notation you may need.

2.23 Put the medical decision making problem in section 2.3.6 into the format of the universal modeling framework in section 2.2. Introduce and define any additional notation you may need.

2.24 Put the scientific exploration problem in section 2.3.7 into the format of the universal modeling framework in section 2.2. Introduce and define any additional notation you may need.

Diary problem

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

2.25 Which of the canonical problems (you may name more than one) seem to use the language that best fits your diary problem.