REINFORCEMENT LEARNING AND STOCHASTIC OPTIMIZATION

A unified framework for sequential decisions

Warren B. Powell

July 7, 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 11

DESIGNING POLICIES

Now that we have learned how to simulate an exogenous process $W_1, \ldots, W_t, \ldots$, we return to the challenge of finding a policy that solves our objective function from chapter 9

$$\max_{\pi \in \Pi} \mathbb{E} \left\{ \sum_{t=0}^{T} C_t(S_t, X_t^\pi(S_t)) | S_0 \right\}. \tag{11.1}$$

This objective function as been the basis of our "model first, then solve" approach. But now it is time to solve. This leaves us with the question: How in the world do we search over some arbitrary class of policies?

This is precisely the reason that this form of the objective function is popular with mathematicians who do not care about computation, or in communities where it is already clear what type of policy is being used. However, equation (11.1) is not widely used, and we believe the reason is that there has not been a natural path to computation. In fact, entire fields have emerged which focus on particular classes of policies.

In this chapter, we address the problem of searching over policies in a general way. Our approach is quite practical in that we organize our search using classes of policies that are widely used either in practice or in the research literature. We start by clarifying one area of confusion, which is the precise meaning of the term "policy" which is popular only in certain communities. A simple definition of a policy is:

Definition 11.0.1. *A* **policy** *is a method that determines a decision given the information in state S_t... any method.*

Algorithm	Format	Prejudice
Behavior	Formula	Principle
Belief	Grammar	Procedure
Bias	Habit	Process
Canon	Laws/bylaws	Protocols
Code	Manner	Recipe
Commandment	Method	Ritual
Conduct	Mode	Rule
Control law	Mores	Style
Convention	Orthodoxy	Syntax
Culture	Patterns	Technique
Customs	Plans	Template
Duty	Policies	Tenet
Etiquette	Practice	Tradition
Fashion	Precedent	Way of life

Table 11.1 Words from the English language that describe methods for making decisions.

The "any method" is included to counteract the assumption by many that "policy" refers specifically and narrowly to analytical functions, which is just one of our four classes of policies.

"Policies" arise in so many settings in human behavior that it should not be surprising that there are many words that have the same meaning. Table 11.1 provides a number of examples.

The problem with the concept of a policy is that it refers to *any* method for determining a decision given a state, and as a result it covers a wide range of algorithmic strategies, each suited to different problems with different computational requirements. While we avoided the use of policies in chapter 4 for problems where we could compute the expectation exactly (or a sampled approximation of the expectation), the concept of designing policies pervades any adaptive learning algorithm (whether it is derivative-based of derivative-free), as well as the entire family of problems with state-dependent costs and constraints.

We are going to start by describing a spectrum of problems ranging from (deterministic) optimization to machine learning, and then we are going to contrast our problem of searching for the best policy to the search problems that these other problem areas pose.

Then, we are going to return to our four classes of policies that we first introduced in chapter 1 and then again in more depth in chapter 7. The remainder of the chapter is designed to help with the search for the best policy for a particular application. This chapter then sets up the more in-depth presentations on each of the four classes provided in chapters 12 - 19.

11.1 FROM OPTIMIZATION TO MACHINE LEARNING TO SEQUENTIAL DECISION PROBLEMS

If we have a linear programming problem, anyone with training in deterministic optimization would write down a model that looks like

$$\min_x c^T x$$

subject to

$$\begin{aligned} Ax &= b, \\ x &\geq 0. \end{aligned}$$

In real applications, the challenge is creating the A-matrix, but this process is well understood, and there are computer packages that can take these models and solve them, even when x is a vector with thousands, even hundreds of thousands, of variables (or dimensions).

Just as popular is the format used for deterministic optimal control, where we have to manage a system over time by choosing a set of controls $u_0, u_1, \ldots, u_T$ (imagine the forces on a vehicle such as landing a SpaceX rocket) to minimize a loss function $L(x_t, u_t)$ when the system is in "state" x_t (for example, the location and velocity of our rocket). The canonical control problem would be written

$$\min_{u_0,\ldots,u_T} \sum_{t=0}^{T} L(x_t, u_t), \tag{11.2}$$

where the state x_t (this is standard notation in this community) evolves according to a *transition function* which is written

$$x_{t+1} = f(x_t, u_t). \tag{11.3}$$

The controls may be subject to constraints. Again, there are standard packages for solving versions of this problem.

A different problem that is very relevant to our work arises in machine learning, where we want to find a function (typically called a "statistical model") $f(x|\theta)$, that minimizes the error between observed inputs x^n and the corresponding output y^n for a training dataset $(x^n, y^n), n = 1, \ldots, N$. For example, a linear model would be written

$$y = \theta_0 + \theta_1\phi_1(x) + \theta_2\phi_2(x) + \ldots + \varepsilon, \tag{11.4}$$

where $\phi_f(x)$ is a feature of the input data x. Let $f \in \mathcal{F}$ be a family of functions (models), where f might specify the structure (such as the linear model in (11.4)) and the features $(\phi_f(x))$. Next let $\theta \in \Theta^f$ be the tunable parameters associated with model f. Our optimization problem is to find the best function (model), and the best parameters θ associated with the function, a problem we write as

$$\min_{f\in\mathcal{F},\theta\in\Theta^f} \sum_{n=1}^{N} (y^n - f(x^n|\theta))^2. \tag{11.5}$$

Here we see an optimization problem written in terms of optimizing over functions, along with any parameters for that function. For machine learning applications, $\mathcal{F}$ covers lookup tables, parametric models and nonparametric models, and all the choices within these sets (as we covered in chapter 3).

These models are very standard. Readers trained in any of these fields would recognize these models, and would have access to software libraries designed to solve them. These modeling languages are spoken around the world.

The optimization for sequential decision problems, given by equation (11.1), involves searching over policies, which parallels the search over functions in machine learning

("policies" are all examples of functions). However, policies span a much wider range of functions. For example, we are going to see that the first of our four classes of policies include every class of function that we might consider in machine learning.

In this chapter, we are going to revisit these four classes (which we first saw in chapters 1, 4, and 7) in greater depth. The hope is that after finishing this chapter, a reader looking to solve a particular problem might have an idea of which one (or two) classes of policies might be best suited for a particular problem. Then we are going to spend chapters 12 - 19 looking into the four classes in even more detail.

11.2 THE CLASSES OF POLICIES

There are two fundamental strategies for creating policies, each of which can be further divided into two classes, creating our four classes of policies. The two strategies are given by

Policy search - Here we are using equation (11.1) directly to search over a) classes of functions and b) parameters that characterize a particular class of function.

Lookahead approximations - These are policies that approximate (sometimes exactly) the downstream value of an action taken now.

Both of these can lead to optimal policies under certain circumstances, but only in special cases where we can exploit structure. Since these are relatively rare, a variety of approximation strategies have evolved.

Policy search is based on the principle of assuming that the policy $X^\pi(S_t|\theta)$ belongs to some class of functions, which are typically parametric, but may be nonparametric (that is, locally parametric). Let the set $f \in \mathcal{F}$ capture the structure of the function, and let $\theta \in \Theta^f$ be the tunable parameters associated with each function. The design of the set $\mathcal{F}$ and the choice of $f \in \mathcal{F}$ is often (not always) more art than science. We let $\pi = (f \in \mathcal{F}, \theta \in \Theta^f)$ describe both the type of function and the parameters.

The policy search problem can be written generally as

$$\max_{\pi=(f\in\mathcal{F},\theta\in\Theta^f)} \mathbb{E}_{S_0}\mathbb{E}_{W_1,\ldots,W_T|S_0}\left\{\sum_{t=0}^{T} C(S_t, X^\pi(S_t|\theta))|S_0\right\}. \tag{11.6}$$

Note that we can use any of our family of objective functions (cumulative reward, final reward) and uncertainty operators (described in section 9.8.4) such as expectation (the most common), max-min (robust optimization), or any of the risk measures that emphasize the tails of the distribution.

There are two class of policies within the policy search class:

Policy function approximations (PFAs) - These are analytical functions that map a state to a feasible action. These functions can be any of the three classes of functions we introduced in chapter 3:

Lookup tables - Also referred to as tabular functions, lookup tables mean that we have a discrete decision $X^\pi(S)$ for each discrete state S.

Parametric representations - These are explicit, analytical functions for $X^\pi(S)$ which generally involve a vector of parameters that we typically represent by

θ. Thus, we might write our policy as

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

where $\phi_f(S),\ f \in \mathcal{F}$ is a set of features tuned for approximating the value function or the policy. Neural networks are a class of parametric functions (see section 3.9.3) that are popular in the engineering controls community, where they may be used to approximate either the policy or the value function.

Nonparametric representations - Nonparametric representations offer a more general way of representing functions, but at a price of greater complexity.

PFAs are typically limited to discrete actions, or low-dimensional (and typically continuous) vectors. PFAs are discussed in depth in chapter 12. Note that PFAs include all the classes of statistical models such as those reviewed in chapter 3.

Cost function approximations (CFAs) - These are parameterized optimization models where we may use a parameterized modification of the objective function, subject to a (possibly parameterized) approximation of the constraints. CFAs are optimization problems which could be a simple as a sort (such as the UCB policies introduced in chapter 7), or it could involve solving large linear or integer programs such as scheduling an airline or planning a supply chain. CFAs have the general form

$$X^{CFA}(S_t|\theta) = \arg\max_{x\in\mathcal{X}_t(\theta)} \bar{C}_t(S_t, x|\theta),$$

where $\bar{C}_t(S_t, x|\theta)$ is a parametrically modified cost function, subject to a parametrically modified set of constraints. CFAs are covered in chapter 13.

Lookahead policies are based on trying to solve what will at first look like a rather frightening expression:

$$X_t^*(S_t) = \arg\max_{x_t}\left(C(S_t,x_t) + \mathbb{E}\left\{\max_{\pi}\mathbb{E}\left\{\sum_{t'=t+1}^{T} C(S_{t'}, X_{t'}^{\pi}(S_{t'}))\,\middle|\, S_{t+1}\right\}\middle|\, S_t, x_t\right\}\right). \quad (11.7)$$

It should not come as a surprise that we cannot compute this, so we turn to approximations. There are two broad classes of approximation strategies, which are given by:

Value function approximations (VFAs) - These are policies based on an approximation of the value of being in a state. These have the general form

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

where we are using the post-decision state variable that we introduced in section 9.4.5, which allows us to compute our policy without an imbedded expectation. VFAs represent a rich and challenging algorithmic strategy that we cover in chapters 14-18.

Direct lookahead policies (DLAs) - This last class of policies directly solves an approximate version of the lookahead policy in equation (11.6). Direct lookahead policies are covered in chapter 19.

Combined, these create four classes of policies (more precisely, these are meta-classes) that encompass every algorithmic strategy that has been proposed for any sequential stochastic optimization problem. We claim that these classes cover any heuristic methods already used in practice, as well as everything covered in the research literature.

Some observations:

- The first three classes of policies (PFAs, CFAs and VFAs) introduce four different types of functions we might approximate (we first saw these in chapter 3). These include 1) approximating the function we are maximizing $\mathbb{E}F(x,W)$, 2) the policy $X^\pi(S)$, 3) the objective function or constraints, or 4) the downstream value of being in a state $V_t(S_t)$. Function approximation plays an important role in stochastic optimization, and this brings in the disciplines of statistics and machine learning.

- The class of functions in the PFA class is precisely the set of three classes of approximating architectures from machine learning: lookup tables, parametric, and nonparametric. The only difference between machine learning and searching for the best PFA policy is the objective function. Machine learning uses a training dataset $(x^n, y^n), n = 1, \ldots, N$ to solve

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

  which requires a training dataset. Policy search requires a performance metric $C(S,x)$, and a model (the transition function $S^M(s,x,W)$) to create the objective function in equation (11.1).

- The last three classes of policies (CFAs, VFAs and DLAs) all use an imbedded $\arg\max$ (or $\arg\min$) which means we have to solve a maximization problem as a step in computing the policy. This maximization (or minimization) problem may be fairly trivial (for example, sorting the value of a set of choices), or quite complex (some applications require solving large integer programs).

- It is possible to get very high quality results from relatively simple policies if we are allowed to tune them (these would fall under policy search). However, this opens the door to using relatively simple lookahead policies (for example, using a deterministic lookahead) which has been modified by tunable parameters for helping to manage uncertainty.

These four classes of policies encompass all the disciplines that we reviewed in chapter 2. We started to hint at the full range of policies in chapter 7 when we addressed derivative-free stochastic optimization. We are going to cover these policies in considerably more depth over chapters 12 - 19. Our goal is to provide a foundation for designing effective policies for the full modeling framework we introduced in chapter 9.

In the remainder of this chapter, we describe these policies in somewhat more depth, but defer to later chapters for complete descriptions. Skimming this chapter is the best way to get a sense of all four classes of policies. We use an energy storage application in section 11.9 to demonstrate that each of these four classes may work best on the same problem class, depending on the specific characteristics of the data.

11.3 POLICY FUNCTION APPROXIMATIONS

It is often the case that we have a very good idea of how to make a decision, and we can design a function (which is to say a policy) that returns a decision which captures the structure of the problem. For example:

EXAMPLE 11.1

A policeman would like to give tickets to maximize the revenue from the citations he writes. Stopping a car requires about 15 minutes to write up the citation, and the fines on violations within 10 miles per hour of the speed limit are fairly small. Violations of 20 miles per hour over the speed limit are significant, but relatively few drivers fall in this range. It is clear that the best policy will be to choose a speed, say θ^{speed}, above which he writes out a citation. The problem is choosing θ^{speed}.

EXAMPLE 11.2

A utility wants to maximize the profits earned by storing energy in a battery when prices are lowest during the day, and releasing the energy when prices are highest. There is a fairly regular daily pattern to prices. The optimal policy can be found by solving a dynamic program or stochastic lookahead policy, but it is fairly apparent that the policy is to charge the battery at one time during the day, and discharge it at another. The problem is identifying these times.

EXAMPLE 11.3

A trader likes to invest in IPOs, wait a few days and then sell, hoping for a quick bump. She wants to use a rule of waiting d days at which point she sells. The problem is to determine d.

EXAMPLE 11.4

A drone can be controlled using a series of actuators that govern the force applied in each of three directions to control acceleration, speed and location (in that order). The logic for specifying the force in each direction can be controlled by a neural network which has to be tuned to produce the best results.

EXAMPLE 11.5

We are holding a stock, and would like to sell it when it goes over a price θ^{sell}. How should we determine θ^{sell}?

EXAMPLE 11.6

In an inventory policy, we will order new product when the inventory S_t falls below θ^{min}. When this happens, we place an order $x_t = \theta^{max} - S_t$, which means we "order up to" θ^{max}. We need to determine $\theta = (\theta^{min}, \theta^{max})$.

EXAMPLE 11.7

We might choose to set the output x_t from a water reservoir, as a function of the state (the level of the water) S_t of the reservoir, using a linear function of the form $x_t = \theta_0 + \theta_1 S_t$. Or we might desire a nonlinear relationship with the water level, and use a basis function $\phi(S_t)$ to produce a policy $x_t = \theta_0 + \theta_1 \phi(S_t)$.

The most common type of policy function approximation is some sort of parametric model. Imagine a policy that is linear in a set of basis functions $\phi_f(S_t),\ f \in \mathcal{F}$. For example, if S_t is a scalar, we might use $\phi_1(S_t) = S_t$ and $\phi_2(S_t) = S_t^2$. We might also create a constant basis function $\phi_0(S_t) = 1$. Let $\mathcal{F} = \{0, 1, 2\}$ be the set of three basis functions. Assume that we feel that we can write our policy in the form

$$X^\pi(S_t|\theta) = \theta_0\phi_0(S_t) + \theta_1\phi_1(S_t) + \theta_2\phi_2(S_t). \tag{11.8}$$

Here, the index "π" carries the information that the function is linear in a set of basis functions, the set of basis functions, and the parameter vector θ. Policies with this structure are known as *linear decision rules* or, if you want to sound fancy, *affine policies*, because they are linear in the parameter vector θ.

The art is coming up with the structure of the policy. The science is in choosing θ, which we do by solving the stochastic optimization problem

$$\max_\theta F^\pi(\theta) = \mathbb{E}\sum_{t=0}^{T} \gamma^t C(S_t, X^\pi(S_t|\theta)). \tag{11.9}$$

Here, we write $\max_\theta$ because we have fixed the class of policies, and we are now searching within a well-defined space. If we were to write $\max_\pi \ldots$, a proper interpretation would be that we would be searching over different functions (e.g. different sets of basis functions), or perhaps even different classes, in addition to searching for whatever parameters θ are associated with that class. Note that we will let π be both the class of policy as well as its parameter vector θ, but we still write $F^\pi(\theta)$ explicitly as a function of θ.

The major challenge we face is that we cannot compute $F^\pi(\theta)$ in any compact form, primarily because we cannot compute the expectation. Instead, we have to depend on Monte Carlo samples. Fortunately, there is a field known as stochastic search to help us with this process. We describe these algorithms in more detail in chapter 12, but the work all draws on derivative-based stochastic optimization (chapter 5) and derivative-free stochastic search (chapter 7).

Parametric policies are popular because of their compact form, but are largely restricted to stationary problems where the policy is not a function of time. Imagine, for example, a situation where the parameter vector in our policy (11.8) is time dependent, giving us a policy of the form

$$X_t^\pi(S_t|\theta) = \sum_{f\in\mathcal{F}} \theta_{tf}\phi_f(S_t). \tag{11.10}$$

Now, our parameter vector is $\theta = (\theta_t)_{t=0}^T$, which is generally dramatically larger than the stationary problem. Solving equation (11.9) for such a large parameter vector (which would easily have hundreds or thousands of dimensions) becomes intractable unless we can compute derivatives of $F^\pi(\theta)$ with respect to θ.

We cover policy function approximations, and how to optimize them, in much greater depth in chapter 12.

11.4 COST FUNCTION APPROXIMATIONS

Cost function approximations represent a class of policy that has been largely overlooked in the academic literature, yet it is widely used in industry. In a nutshell, CFAs involve solving a deterministic optimization problem that has been modified so that it works well over time, under uncertainty.

To illustrate, we might start with a myopic policy of the form

$$X_t^{Myopic}(S_t) = \arg\max_{x \in \mathcal{X}_t} C(S_t, x), \tag{11.11}$$

where $\mathcal{X}_t$ captures the set of constraints. We emphasize that x may be high-dimensional, with a linear cost function such as $C(S_t, x) = c_t x$, subject to a set of linear constraints:

$$\begin{aligned} A_t x_t &= b_t, \\ x_t &\leq u_t, \\ x_t &\geq 0. \end{aligned}$$

This hints at the difference in the type of problems we can consider with CFAs. A sample application might involve assigning resources (people, machines) to jobs (tasks, orders) over time. Let c_{trj} be the cost (or contribution) of assigning resource r to job j at time t, where c_t is the vector of all assignment costs. Also let $x_{trj} = 1$ if we assign resource r to job j at time t, 0 otherwise. Our myopic policy, which assigns resources to jobs to minimize costs now, may perform reasonably well. Now assume that we would like to see if we could make it work a little better.

We can sometimes improve on a myopic policy by solving a problem with a modified objective function.

$$X_t^{CFA}(S_t|\theta) = \arg\max_{x \in \mathcal{X}_t} \big(C(S_t, x) + \underbrace{\sum_{f \in \mathcal{F}} \theta_f \phi_f(S_t, x)}_{\text{Cost function correction term}}\big). \tag{11.12}$$

The new term in the objective is called a "cost function correction term."

More often (in our experience) we work by modifying the constraints. We might use

$$\begin{aligned} A_t x_t &= \theta^1 \odot b_t + \theta^2, \\ x_t &\leq u_t - \theta^3, \\ x_t &\geq \theta^4. \end{aligned}$$

Here, the operator $\odot$ means that we multiple the ith element of θ^1 times the ith element of b_t. θ^1 and θ^2 are assumed to be the same dimension as the vector b_t, while θ^3 and θ^4 are each assumed to be the same dimension as x_t. The parameter θ^3 can be used to shrink the capacity of storage batteries so we have spare capacity to store a burst of energy from wind, while θ^4 might be used to ensure safety stocks in a supply chain problem.

There will be times when we might scale the matrix A_t. For example, airlines have to insert schedule slack to handle possible weather delays. Instead of using the average flight time between two cities, an airline may use the $80th$ percentile, which of course is a tunable parameter.

In chapter 13, we discuss a wider range of approximation strategies, including modified constraints and hybrid lookahead policies.

11.5 VALUE FUNCTION APPROXIMATIONS

The next class of policy is based on approximating the value of being in a state resulting from an action we take now. The core idea starts with Bellman's optimality equation (that we first saw in chapter 2 but study in much greater depth in chapter 14), which is written

$$V_t(S_t) = \max_{x \in \mathcal{X}_t} \big(C(S_t, x) + \gamma \mathbb{E}\{V_{t+1}(S_{t+1})|S_t\}\big). \tag{11.13}$$

where $S_{t+1} = S^M(S_t, x, W_{t+1})$. If we use the post-decision state variable S_t^x,

$$V_t(S_t) = \max_{x \in \mathcal{X}_t} \big(C(S_t, x) + V_t^x(S_t^x)\big), \tag{11.14}$$

where $V_t^x(S_t^x)$ is the (optimal) value of being in post-decision state S_t^x at time t. Chapter 14 deals with problems where Bellman's equation (11.13) (or the post-decision form in (11.14)) can be computed exactly. For the vast range of problems where this is not possible, we can try to replace the value function with some sort of statistical approximation we call $\overline{V}_t(S_t)$.

In later chapters (16 - 18), we are going to address the difficult challenge of estimating value function approximations. Given a VFA, we can quickly create a VFA-based policy. If we use the pre-decision value function, we obtain

$$X_t^{VFA-pre}(S_t) = \arg\max_x \left(C(S_t, x) + \gamma \mathbb{E}\left\{\overline{V}_{t+1}(S_{t+1})|S_t\right\}\right), \tag{11.15}$$

where $S_{t+1} = S^M(S_t, x, W_{t+1})$. Often the expectation is problematic since in many settings W_{t+1} is multidimensional. We could overcome this using a sampled model, where we would compute

$$X_t^{VFA-pre}(S_t) = \arg\max_x \left(C(S_t, x) + \gamma \frac{1}{|\hat{\Omega}_{t+1}|} \sum_{\omega \in \hat{\Omega}_t} \overline{V}_{t+1}(S_{t+1}(\omega))\right), \tag{11.16}$$

where $S_{t+1}(\omega) = S^M(S_t, x, W_{t+1}(\omega))$. If there is a natural (and ideally more compact) post-decision state, then

$$X_t^{VFA-post}(S_t) = \arg\max_x \left(C(S_t, x) + \gamma \overline{V}_t^x(S_t^x)\right), \tag{11.17}$$

where the post-decision state $S_t^x = S^{M,x}(S_t, x)$ is a deterministic function of the pre-decision state S_t and decision x_t. Clearly, if we can take advantage of a post-decision state, then the post-decision version of a VFA policy in (11.17) is the easiest to use.

Although dynamic programming is most often used in settings with discrete actions, we can handle vector-valued decisions x_t for problems where the contribution function $C(S_t, x_t)$ is concave in x_t, which produces concave value functions. Chapter 18 shows how to create value function approximations that exploit this property, making it possible to solve very high-dimensional resource allocation problems.

A closely related policy, developed under the umbrella of reinforcement learning within computer science, is to use Q-factors which approximate the value of being in a state S_t and taking discrete action a_t (the strategy only works for discrete actions). Let $\bar{Q}^n(s, a)$ be our approximate value of being in state s and taking action a after n iterations. Q-learning uses some rule to choose a state s^n and action a^n, and then uses some process to simulate

a subsequent downstream state s' (which might be observed from a physical system). It then proceeds by computing

$$\hat{q}^n(s^n, a^n) = C(s^n, a^n) + \max_{a'} \bar{Q}^{n-1}(s', a'), \tag{11.18}$$

$$\bar{Q}^n(s^n, a^n) = (1-\alpha)\bar{Q}^{n-1}(s^n, a^n) + \alpha\hat{q}^n(s^n, a^n). \tag{11.19}$$

Given a set of Q-factors $\bar{Q}^n(s, a)$, the policy is given by

$$A^\pi(S_t) = \arg\max_a \bar{Q}^n(S_t, a). \tag{11.20}$$

Q-learning became quite popular largely because of its simplicity, but there is a big gap between coding the basic updates in (11.18)-(11.19), and getting it to actually work. There are a number of algorithmic choices that have to be made, such as how to choose the state s^n and action a^n during the learning process, and how to approximate $Q(s, a)$ when the state space is large (which it always is).

A popular problem for illustrating Q-learning is playing games such as Go, which seem hard, but are relatively trivial compared to the massive range of sequential decision problems that arise in settings such as energy, health, transportation and logistics. For example, try to translate the variations of the energy storage problem in section 9.9 to fit this style. The problem with rolling forecasts (in section 9.9.4) has a state variable with 42 dimensions, most of them continuous.

11.6 DIRECT LOOKAHEAD APPROXIMATIONS

We save direct lookahead policies for last because this is the most brute-force approach among the four classes of policies. A good description for DLA policies is that they are the class you turn to when all else fails, and all else often fails.

11.6.1 The basic idea

Imagine that we are in a state S_t. We would like to choose an action x_t that maximizes the contribution $C(S_t, x_t)$ now, plus the value of the state that our action takes us to. Given S_t and x_t, we will generally experience some randomness W_{t+1} that then takes us to state S_{t+1}. The value of being in state S_{t+1} is given by

$$\begin{aligned} V^*_{t+1}(S_{t+1}) &= \max_\pi \mathbb{E}\left\{\sum_{t'=t+1}^{T} C(S_{t'}, X^\pi_{t'}(S_{t'}))|S_{t+1}\right\}. \\ &= \mathbb{E}\left\{\sum_{t'=t+1}^{T} C(S_{t'}, X^*_{t'}(S_{t'}))|S_{t+1}\right\}. \end{aligned} \tag{11.21}$$

We could write our optimal policy just as we did above in equation (11.13)

$$X^*(S_t) = \arg\max_{x_t} \big(C(S_t, x_t) + \mathbb{E}\{V^*_{t+1}(S_{t+1})|S_t, x_t\}\big),$$

but now we are going to recognize that we generally cannot compute the optimal value function $V^*_{t+1}(S_{t+1})$. Rather than try to approximate this function, we are going to substitute in the definition of $V^*_{t+1}(S_{t+1})$ from (11.21), which gives us

$$X^*_t(S_t) = \arg\max_{x_t} \left(C(S_t, x_t) + \mathbb{E}\left\{\mathbb{E}\left\{\sum_{t'=t+1}^{T} C(S_{t'}, X^*_{t'}(S_{t'}))\,\middle|\, S_{t+1}\right\}|S_t, x_t\right\}\right). \tag{11.22}$$

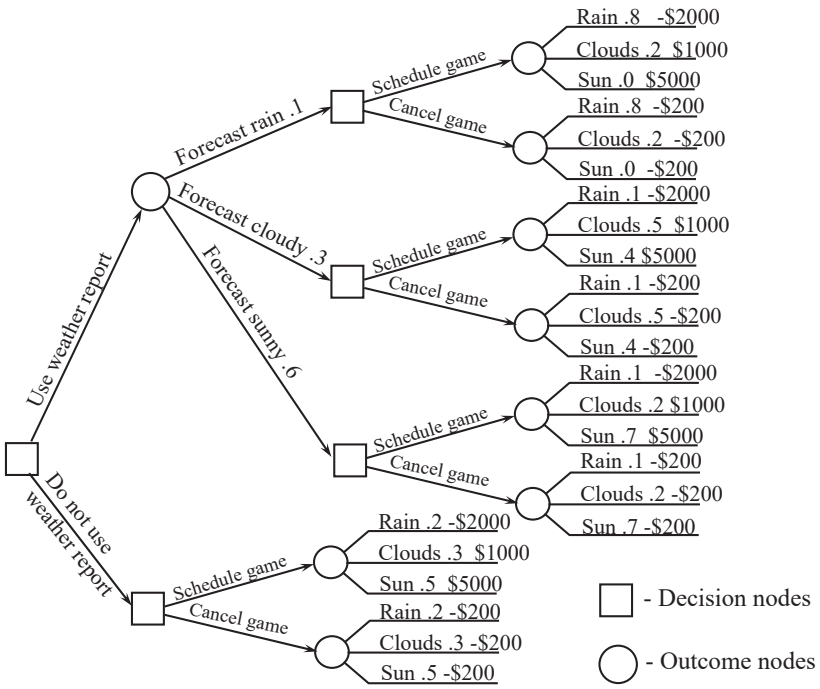


Figure 11.1 Decision tree showing decision nodes and outcome nodes for the setting of deciding whether to schedule a baseball game.

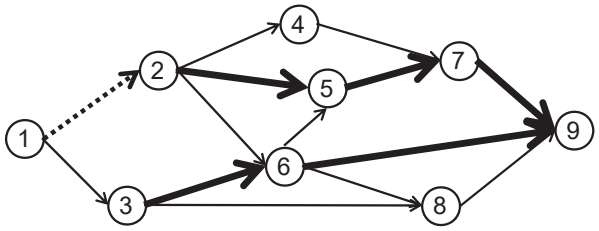
Another way of writing (11.22) is to explicitly imbed the search for the optimal policy in the lookahead portion, giving us

$$X_t^*(S_t) = \arg\max_{x_t} \left(C(S_t, x_t) + \mathbb{E} \left\{ \max_{\pi} \mathbb{E} \left\{ \sum_{t'=t+1}^{T} C(S_{t'}, X_{t'}^{\pi}(S_{t'})) \middle| S_{t+1} \right\} | S_t, x_t \right\} \right). \quad (11.23)$$

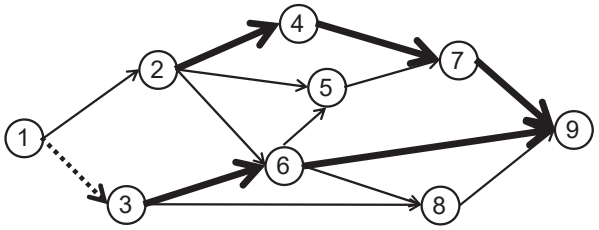
Equation (11.23) can look particularly daunting, until we realize that this is exactly what we are doing when we solve a decision tree (exercise 11.10 provides a numerical example) which is illustrated in figure 11.1. Remember that a "decision node" in a decision tree (the squares) corresponds to the state S_t (if we are referring to the first node), or the states $S_{t'}$ for the later nodes.

We could use some generic rule $X_{t'}^{\pi}(S_{t'})$ for making a decision, or we can solve the decision tree by stepping backward through the tree to find the optimal action $x_{t'}^*$ for each discrete state $S_{t'}$, which is a lookup table representation for the optimal policy $X_{t'}^*(S_{t'})$. We just have to recognize that $X_{t'}^{\pi}(S_{t'})$ refers to *some* rule for choosing an action out of node $S_{t'}$, while $X_{t'}^*(S_{t'})$ is the best action out of node $S_{t'}$.

To parse equation (11.23), the first expectation, which is conditioned on state S_t and action x_t, is over the first set of random outcomes out of the circle nodes. The inner $\max_\pi$ refers generally to the process of finding the best action out of *each* of the remaining decision nodes, before knowing the downstream random outcomes. We then evaluate this policy by taking the expectation over all outcomes.



(a)



(b)

Figure 11.2 (a) Decision to go (1,2) given the path 2-5-7-9. (b) Decision to go (1,3) when path out of node 2 changes.

Another way to help understand equation (11.22) (or (11.23)) is to think about a deterministic shortest path problem. Consider the networks shown in figure 11.2. If we know that we would use the path 2-5-7-9 to get from 2 to 9, we would choose to go from 1 to 2 to take advantage of this path. But if we elect to use a different path out of node 2 (a costlier path), then our decision from node 1 might be to go to node 3. The decisions we are thinking about making downstream can affect the decision we make now.

This key insight translates to the world of uncertainty with a twist: the decision we make now depends on the *policy* we use to make decisions in future. While we dream about using optimal policies, in practice that is just a dream. Chapter 19 explores the idea of using simpler policies in our lookahead model to help streamline computation.

11.6.2 Modeling the lookahead problem

This hints at one of the most popular ways of approximating the future for a stochastic problem, which is simply to use a deterministic approximation of the future. We can create what we are going to call a *deterministic lookahead model*, where we act as if we are optimizing in the future, but only for an approximate model.

So we do not confuse the lookahead model with the model we are trying to solve, we are going to introduce two notational devices. First, we are going to use tilde's for state, decision variables and exogenous information variables. Second, we are going to index them by t and t', where t refers to the time at which we are making a decision, and t' indexes time within our lookahead model. Thus, a deterministic lookahead model over a

horizon $t, \ldots, t+H$, would be formulated as

$$X_t^{LA-Det}(S_t|\theta) = \arg \max_{x_t,(\tilde{x}_{t,t+1},\ldots,\tilde{x}_{t,t+H})} \left(C(S_t, x_t) + \sum_{t'=t+1}^{t+H} C(\tilde{S}_{tt'}, \tilde{x}_{tt'}) \right). \quad (11.24)$$

Here, we have replaced the model of the problem from time $t+1$ to the end of horizon T with a deterministic approximation that goes out to some truncated horizon $t+H$.

There are special cases where we can solve a stochastic lookahead model. One is problems with small numbers of discrete actions, and relatively simple forms of uncertainty. In this case, we can represent our problem using a decision tree such as the one we illustrated in figure 11.1. A decision tree allows us to find the best decision for each node (that is, each state), which is a form of lookup table policy. The problem is that decision trees explode in size for most problems, limiting their usefulness. In chapter 19, we describe methods for formulating and solving stochastic lookahead models using Monte Carlo methods.

While this is the simplest type of lookahead policy, it illustrates the basic idea. We cannot solve the true problem in (11.23), so we introduced a variety of approximations. Deterministic lookahead models tend to be relatively easy to solve (but not always). However, using a deterministic approximation of the future means that we may make decisions now that do not properly prepare us for random events that may happen in the future. Thus, there is considerable interest in solving a lookahead model that recognizes that the future is uncertain.

The design of lookahead models is as much art as science. We can simplify the lookahead model using strategies such as limiting the horizon, using a sample of random outcomes, discretization, ignoring the updating of selected variables, and using simplified policies for the lookahead model. We note that there are entire books dedicated to specific ways of approximating lookahead models.

Direct lookahead policies are covered in considerably greater depth in chapter 19. For now, we are going to provide a peek inside the design of policies in the lookahead model which we sometimes call the "policy-within-a-policy."

11.6.3 The policy-within-a-policy

As a hint of what we mean by designing the "policy-with-a-policy," we might choose to use a linear decision rule that we could write as

$$\tilde{X}_t^{Lin}(\tilde{S}_{tt'}|\theta_t) = \theta_{t0} + \theta_{t1}\phi_1(\tilde{S}_{tt'}) + \theta_{t2}\phi_2(\tilde{S}_{tt'}).$$

We could then write our stochastic lookahead policy as

$$X_t^{LA-Stoch}(S_t) = \arg\max_{x_t} \Bigg(C(S_t, x_t) + \tilde{E}\left\{ \max_{\tilde{\theta}_t} \tilde{E}\left\{ \sum_{t'=t+1}^{T} C(\tilde{S}_{tt'}, \tilde{X}_t^{Lin}(\tilde{S}_{tt'}|\tilde{\theta}_t))|\tilde{S}_{t,t+1} \right\} |S_t, x_t \right\} \Bigg). \quad (11.25)$$

Keep in mind that when you see the expectation operator $\tilde{E}$, this means taking an expectation over the approximate stochastic model. In fact, it almost always means taking an expectation over a sampled model, so just think of taking a sample of whatever random variable is involved. The first $\tilde{E}$ is taking an expectation over the first set of exogenous outcomes,

$\widetilde{W}_{t,t+1}$, while the second $\tilde{E}$ requires sampling the entire sequence $\widetilde{W}_{t,t+2}, \ldots, \widetilde{W}_{t,t+H}$. Given this sample, we now simulate our lookahead policy $\tilde{X}_t^{Lin}(\tilde{S}_{tt'}|\theta_t)$ (often called a *rollout policy*) to help us estimate the downstream effects from making decision x_t now.

11.7 HYBRID STRATEGIES

Now that we have identified the four major (meta)classes of policies, we need to recognize that we can also create hybrids by mixing the different classes.

The set of (possibly tunable) policy function approximations, parametric cost function approximations, value function approximations and direct lookahead policies represent the core tools in the arsenal for finding effective policies for sequential decision problems. Given the richness of applications, it perhaps should not be surprising that we often turn to mixtures of these strategies.

Cost function approximation with policy function approximations

A major strength of a deterministic lookahead policy is that we can use powerful math programming solvers to solve high-dimensional deterministic models. A challenge is handling uncertainty in this framework. Policy function approximations, on the other hand, are best suited for relatively simple decisions, and are able to handle uncertainty by capturing structural properties (when they can be clearly identified). PFAs can be integrated into high-dimensional models as nonlinear penalty terms acting on individual (scalar) variables.

As an example, consider the problem of assigning resources (imagine we are managing blood supplies) to tasks, where each resource is described by an attribute vector a (the blood type and age) while each task is described by an attribute vector b (the blood type of a patient, along with other attributes such as whether the patient is an infant or has immune disorders). Let c_{ab} be the contribution we assign if we assign a resource of type a to a patient with blood type b. Let R_{ta} be the number of units of blood type a available at time t, and let D_{tb} be the demand for blood b. Finally let x_{tab} be the number of resources of type a assigned to a task of type b. A myopic policy (a form of cost function approximation) would be to solve

$$X^{CFA}(S_t) = \arg\max_{x_t} \sum_{a\in\mathcal{A}} \sum_{b\in\mathcal{B}} c_{ab} x_{tab}. \tag{11.26}$$

subject to

$$\sum_{b\in\mathcal{B}} x_{tab} \leq R_{ta}, \tag{11.27}$$

$$\sum_{a\in\mathcal{A}} x_{tab} \leq D_{tb}, \tag{11.28}$$

$$x_{tab} \geq 0. \tag{11.29}$$

This policy would maximize the total contribution for all blood assignments, but might ignore issues such as a doctor's preference to avoid using blood that is not a perfect match for infants or patients with certain immune disorders.

A doctor's preferences might be expressed through a set of patterns ρ_{ab} which gives the fraction of demand of type b to be satisfied with blood of type a, where $\sum_a \rho_{ab} = 1$. The

vector $\rho_{\cdot b} = (\rho_{ab})_{a \in \mathcal{A}}$ can be viewed as a probabilistic policy describing how to satisfy a demand for a unit of blood of type b (it is a form of PFA).

A natural question would be: why do we need the optimization model? Why can't we just use the patterns ρ_{ab}? The reason is that our patterns might specify how much demand of type b should be supplied with blood with attribute a (we could turn these probabilities around), but in reality we have to balance across all the blood types and demands, which is a much higher dimensionality problem.

The optimization problem described by equations (11.26) - (11.29) easily handles very high dimensional problems. In fact, we can include blood attributes of blood type (8), age (6), whether it is frozen or not (2) and, if we like, the location of the blood (this could number in the hundreds to many thousands). This means that our number of blood attributes could range from 100 to 1 million. Problems of this size easily fall in the scope of modern solvers.

We can combine the high-dimensional capabilities of the optimization model given by equations (11.26) - (11.29) with the low dimensional patterns ρ_{ab} that capture the behavior of the blood management system. These can be combined in a hybrid that would be written

$$X^{CFA-PFA}(S_t|\theta) = \arg\max_{x_t} \sum_{a \in \mathcal{A}} \sum_{b \in \mathcal{B}} \left(c_{ab} x_{tab} + \theta (x_{tab} - D_{tb}\rho_{ab})^2 \right),$$

where θ is a tunable parameter that controls the weight placed on the PFA. This can now be optimized using policy search methods.

Lookahead policies with value function approximations

Deterministic rolling horizon procedures offer the advantage that we can solve them optimally, and if we have vector-valued decisions, we can use commercial solvers. Limitations of this approach are a) they require that we use a deterministic view of the future and b) they can be computationally expensive to solve (pushing us to use shorter horizons). By contrast, a major limitation of value function approximations is that we may not be able to capture the complex interactions that are taking place within our optimization of the future.

An obvious strategy is to combine the two approaches. For low-dimensional action spaces, we can use tree search or a roll-out heuristics for H periods, and then use a value function approximation. If we are using a rolling horizon procedure for vector-valued decisions, we might solve

$$X^{\pi}(S_t) = \arg \max_{x_t, \ldots, x_{t+H}} \sum_{t'=t}^{t+H-1} C(S_{t'}, x_{t'}) + \overline{V}_{t+H}(S_{t+H}),$$

where S_{t+H} is determined by X_{t+H}. In this setting, $\overline{V}_{t+H}(S_{t+H})$ would have to be some convenient analytical form (linear, piecewise linear, nonlinear in S_{t+H}) in order to be used in an appropriate solver.

The hybrid strategy makes it possible to capture the future in a very precise way for a few time periods, while minimizing truncation errors by terminating the tree with an approximate value function. This is a popular strategy in computerized chess games, where a decision tree captures all the complex interactions for a few moves into the future. Then, a simple point system capturing the pieces lost is used to reduce the effect of a finite horizon.

We touch on this stategy only briefly here, but it is arguably one of the most powerful new algorithmic technologies to emerge in stochastic optimization for problems that call

for a lookahead policy (of which there are many). We revisit this strategy in more depth in chapter 13.

We note that recent breakthroughs in the use of computers to solve chess or the Chinese game of Go use a hybrid strategy that mixes lookahead policies (using tree search methods we describe in chapter 19), PFAs (basically rules of how to behave based on patterns derived from looking at past games), and VFAs.

Lookahead policies with cost function approximations

A rolling horizon procedure using a deterministic forecast is, of course, vulnerable to the use of a point forecast of the future. For example, we might be planning inventories for our supply chain for iPhones, but a point forecast might allow inventories to drop to zero if this still allows us to satisfy our forecasts of demand. This strategy would leave the supply chain vulnerable if demands are higher than expected, or if there are delivery delays.

This limitation will not be solved by introducing value function approximations at the end of the horizon. It is possible, however, to perturb the forecasts of demands to account for uncertainty. For example, we could inflate the forecasts of demand to encourage holding inventory. We could multiply the forecast of demand $f_{tt'}^D$ at time t' made at time t by a factor $\theta_{t'-t}^D$. This gives us a vector of tunable parameters $\theta_1^D, \ldots, \theta_H^D$ over a planning horizon of length H. Now we just need to tune this parameter vector to achieve good results over many sample paths.

Tree search with rollout heuristic and a lookup table policy

A surprisingly powerful heuristic algorithm that has received considerable success in the context of designing computer algorithms to play games has evolved under the name "Monte Carlo tree search." MCTS uses a limited tree search, which is then augmented by a rollout heuristic assisted by a user-defined lookup table policy. In other words, this is a direct lookahead policy on a stochastic model that mimics solving the original problem, with the restriction that it is only for decision problems with discrete actions.

For example, a computer might evaluate all the options for a chess game for the next four moves, at which point the tree grows explosively. After four moves, the algorithm might resort to a rollout heuristic (which is a general term implying a simple policy-within-a-policy), assisted by rules derived from thousands of chess games (a form of PFA, similar to our patterns ρ_{ab} above). These rules are encapsulated in an aggregated form of lookup table policy that guides the search for a number of additional moves into the future.

Value function approximation with policy function approximation

Assume we are given a policy $\bar{X}(S_t)$, which might be in the form of a lookup table or a parameterized policy function approximation. This policy might reflect the experience of a domain expert, or it might be derived from a large database of past decisions. For example, we might have access to the decisions of people playing online poker, or it might be the historical patterns of a company. We can think of $\bar{X}(S_t)$ as the decision of the domain expert or the decision made in the field. If the action is continuous, we could incorporate it into our decision function using

$$X^\pi(S_t|\theta) = \arg\max_x \left(C(S_t, x) + \overline{V}(S^{M,x}(S_t, x)) - \theta(\bar{X}(S_t) - x)^2\right).$$

The term $\theta(\bar{X}(S_t) - x)^2$ can be viewed as a penalty for choosing actions that deviate from the external domain expert. The parameter θ controls how important this term is. We note that this penalty term can be set up to handle decisions at some level of aggregation.

Fitting value functions using ADP and policy search

Consider any application of ADP to a problem where we are using a parameterized VFA - linear, nonlinear parametric, or a neural network. We might be playing games, pricing an option, managing energy storage, or solving a high dimensional resource allocation problem.

We can estimate a VFA-like term in two stages. Assume we start with a pure VFA policy using a value function approximation $\overline{V}_t^x(S_t^x|\theta^{VFA})$ around the post-decision state S_t^x using the linear model

$$\overline{V}_t^x(S_t^x|\theta^{VFA}) = \sum_{f\in\mathcal{F}} \theta_f^{VFA}\phi_f(S_t^x), \tag{11.30}$$

where $(\phi_f(S_t^x))_{f\in\mathcal{F}}$ is a user-defined set of features and θ^{VFA} is a set of parameters chosen using approximate dynamic programming algorithms. This gives us a VFA policy that we can write as

$$X_t^{VFA}(S_t|\theta^{VFA}) = \arg\max_x \big(C(S_t,x) + \overline{V}_t^x(S_t^x|\theta^{VFA})\big). \tag{11.31}$$

Chapters 15-17 cover strategies for approximating value functions in much greater depth under the umbrella of approximate dynamic programming (ADP). These methods can produce good solutions, but classical ADP techniques are hardly perfect, especially when using parameterized approximations such as the linear model in equation (11.30). This is the first stage of this hybrid strategy.

For the second stage, we can take our VFA policy $X_t^{VFA}(S_t|\theta^{VFA})$ and, starting with $\theta = \theta^{VFA}$, further tune θ using policy search techniques by solving

$$\max_\theta F(\theta) = \mathbb{E}\sum_{t=0}^{T} C(S_t, X^{VFA}(S_t|\theta)). \tag{11.32}$$

This will typically require the use of one of the algorithms that we introduced in chapters 5 or 7. Let θ^{CFA} be the optimal solution to (11.32). When we use θ^{CFA} in our policy in equation (11.31), it gives us the policy

$$X_t^{CFA}(S_t|\theta^{CFA}) \quad = \quad \arg\max_x \left(C(S_t,x) + \sum_{f\in\mathcal{F}} \theta_f^{CFA}\phi_f(S_t^x)\right).$$

The policy $X_t^{CFA}(S_t|\theta^{CFA})$ is no longer a VFA policy, since there is no reason for $\sum_{f\in\mathcal{F}} \theta_f^{CFA}\phi_f(S_t^x)$ to approximate a value function at this point. The reason is that choosing θ to optimize (11.32) completely loses the objective to make $\sum_{f\in\mathcal{F}} \theta_f^{CFA}\phi_f(S_t^x)$ approximate a value function.

We note in passing that in theory, the CFA-based policy $X_t^{CFA}(S_t|\theta^{CFA})$ should always outperform the VFA-based policy $X_t^{VFA}(S_t|\theta^{VFA})$ since both have the exact same architecture, but the CFA-based policy is tuned specifically to optimize the objective function. There are two reasons why this may not be the case:

- Solving the policy search problem (11.32) introduces noise. The function $F(\theta)$ is often nonconcave, and if the search algorithm is not properly tuned, it can actually end up with a solution that is worse than the starting point.
- The VFA-based policy can easily handle a time-dependent problem, producing a time-dependent policy (in which case we would write θ_t^{VFA} as dependent on time t). The parameters in the CFA-based policy, on the other hand, are assumed to be stationary (that is, they do not depend on time). If they did depend on time, the parameter vector θ_t is now *much* bigger than the stationary parameter vector θ.

Despite these concerns, we believe that starting the stochastic search for the optimization problem in (11.32) using θ^{VFA} as a starting point is likely to produce better results than if we had to use some randomly chosen starting point.

11.8 RANDOMIZED POLICIES

There are several situations where it is useful to randomize a policy:

Exploration-exploitation - This is easily the most common use of randomized policies. Three popular examples of exploration-exploitation policies are:

Epsilon-greedy exploration - This is a popular policy for balancing exploration and exploitation, and can be used for any problem with discrete actions, where the policy has an imbedded $\arg\max_a$ to choose the best discrete action within a discrete set $\mathcal{X} = \{x_1, \ldots, x_M\}$. Let $C(s,x)$ be the contribution from being in state s and taking action x, which might include a value function or a lookahead model. The epsilon-greedy policy chooses an action $x \in \mathcal{X}$ at random with probability ϵ, and chooses the action $\arg\max_{x\in\mathcal{X}} C(s,x)$ with probability $1-\epsilon$.

Boltzmann exploration - Let $\bar{Q}^n(s,x)$ be the current estimate of the value of being in state s and making decision $x \in \mathcal{X} = \{x_1, \ldots, x_M\}$. Now compute the probability of choosing action a according to the Boltzmann distribution

$$P(x|s,\theta) = \frac{e^{\theta \bar{Q}^n(s,x)}}{\sum_{x'\in\mathcal{X}} e^{\theta \bar{Q}^n(s,x')}}.$$

The parameter θ is a tunable parameter, where $\theta = 0$ produces a pure exploration policy, while as θ increases, the policy becomes greedy (choosing the action that appears to be best), which is a pure exploitation policy. The Boltzmann policy chooses what appears to be the best action with the highest probability, but any action may be chosen. This is the reason it is often called a *soft max* operator.

Excitation - Assume that the control x is continuous (and possibly vector-valued). Let Z be a similarly-dimensioned vector of normally distributed random variables with mean 0 and variance 1. An excitation policy perturbs the policy $X^\pi(S_t)$ by adding a noise term such as

$$x_t = X^\pi(S_t) + \sigma Z,$$

where σ is an assumed level of noise.

Thompson sampling - As we saw in chapter 7, Thompson sampling uses a prior on the value of $\mu_x = \mathbb{E}F(x,W)$ is $\mu_x \sim N(\bar{\mu}_x^n, \sigma_x^{2,n})$. Now draw $\hat{\mu}_x^n$ from the distribution $N(\bar{\mu}_x^n, \sigma_x^{2,n})$ for each x, and then choose

$$X^{TS}(S^n) = \arg\max_x \hat{\mu}_x^n.$$

Modeling unpredictable behavior - We may be trying to model the behavior of a system with human input. The policy $X^\pi(S_t)$ may reflect perfectly rational behavior, but a human may behave erratically.

Disguising the state - In a multiagent system, a decision can reveal private information. Randomization can help to disguise private information.

Any of the four classes of policies can be randomized, either by perturbing the decision after it comes out of the policy, or by randomizing inputs such as costs or constraints.

It is possible to convert any randomized policy into a deterministic one by including a uniformly-distributed random variable U_{t-1} (or the normally-distributed variable Z) to the exogenous information process W_{t-1} so that it becomes a part of the state variable S_t. This random variable can then be used to provide the additional information to make $X^\pi(S_t)$ a deterministic function of the (now expanded) state S_t. However, it is standard to refer to the policies above as "random."

11.9 ILLUSTRATION: AN ENERGY STORAGE MODEL REVISITED

In section 9.9, we presented a model of an energy storage problem. We are going to return to this problem and create samples of all four classes of policies, along with a hybrid. We are going to further show that *each* of these policies may work best depending on the data. We recommend reviewing the model since we are going to use the same notation.

11.9.1 Policy function approximation

Our policy function approximation is given by

$$X_t^{PFA}(S_t|\theta) = \begin{cases} x_t^{EL} &= \min\{L_t, E_t\}, \\ x_t^{BL} &= \begin{cases} h_t & \text{If } p_t > \theta^U \\ 0 & \text{If } p_t < \theta^U \end{cases} \\ x_t^{GL} &= L_t - x_t^{EL} - x_t^{BL}, \\ x_t^{EB} &= \min\{E_t - x_t^{EL}, \rho^{chrg}\}, \\ x_t^{GB} &= \begin{cases} \rho^{chrg} - x_t^{EB} & \text{If } p_t < \theta^L \\ 0 & \text{If } p_t > \theta^L \end{cases} \end{cases}$$

where $h_t = \min\{L_t - x_t^{EL}, \min\{R_t, \rho^{chrg}\}\}$. This policy is parameterized by (θ^L, θ^U) which determine the price points at which we charge or discharge.

11.9.2 Cost function approximation

The cost function approximation minimizes a one-period cost plus a tunable error correction term:

$$X^{CFA-EC}(S_t|\theta) = \arg\min_{x_t \in \mathcal{X}_t} \big(C(S_t, x_t) + \theta(x_t^{GB} + x_t^{EB} + x_t^{BL})\big), \quad (11.33)$$

where $\mathcal{X}_t$ captures the constraints on the flows (equations (9.23) - (9.27) are from the model given in section 9.9). We use a linear correction term for simplicity which is parameterized by the scalar θ.

11.9.3 Value function approximation

Our VFA policy uses an approximate value function approximation, which we write as

$$X^{VFA}(S_t) = \arg\min_{x_t \in \mathcal{X}_t} \big(C(S_t, x_t) + \overline{V}_t^x(R_t^x)\big), \tag{11.34}$$

where $\overline{V}_t^x(R_t^x)$ is a piecewise linear function approximating the marginal value of the post-decision resource state. We use methods described in chapter 18 to compute the value function approximation which exploits the natural convexity of the problem. For now, we simply note that the approximation is quite good.

11.9.4 Deterministic lookahead

The next policy is a deterministic lookahead over a horizon H which has access to a forecast of wind energy.

$$X_t^{LA-DET}(S_t) = \arg\min_{(x_t, \tilde{x}_{t+1,t}, \ldots, \tilde{x}_{t,t+H})} \left(C(S_t, x_t) + \sum_{t'=t+1}^{t+H} C(\tilde{S}_{tt'}, \tilde{x}_{tt'}) \right) \tag{11.35}$$

subject to, for $t' = t, \ldots, T$:

$$\tilde{x}_{tt'}^{EL} + \tilde{x}_{tt'}^{EB} \leq f_{tt'}^{E}, \tag{11.36}$$

$$(\tilde{x}_{tt'}^{GL} + \tilde{x}_{tt'}^{EL} + \tilde{x}_{tt'}^{BL}) = f_{tt'}^{L}, \tag{11.37}$$

$$\tilde{x}_{tt'}^{BL} \leq \tilde{R}_{tt'}, \tag{11.38}$$

$$\tilde{x}_{tt'} \geq 0. \tag{11.39}$$

where $f_{tt'}^E$ is the forecast of energy from a wind farm at time t', made at time t, and $f_{tt'}^L$ is a forecast of load (demand) for power. We use tilde's on variables in our lookahead model so they are not confused with the same variable in the base model. The variables are also indexed by t, which is when the lookahead model is formed, and t', which is the time period within the lookahead horizon.

11.9.5 Hybrid lookahead-cost function approximation

Our last policy, $X_t^{LA-CFA}(S_t|\theta^L, \theta^U)$, is a hybrid lookahead with a form of cost function approximation in the form of two additional constraints for $t' = t+1, \ldots, T$:

$$\tilde{R}_{tt'} \geq \theta^L, \tag{11.40}$$

$$\tilde{R}_{tt'} \leq \theta^U. \tag{11.41}$$

These constraints provide buffers to ensure that that we do not plan on the energy level getting too close to the lower or upper limits, allowing us to anticipate that there will be times when the energy from a renewable source is lower, or higher, than we planned. We note that a CFA-lookahead policy is actually a hybrid policy, combining a deterministic lookahead with a cost function approximation (where the approximation is in the modification of the constraints).

11.9.6 Experimental testing

To test our policies, we created five problem variations:

a) A stationary problem with heavy-tailed prices, relatively low noise, moderately accurate forecasts and a reasonably fast storage device.

b) A time-dependent problem with daily load patterns, no seasonalities in energy and price, relatively low noise, less accurate forecasts and a very fast storage device.

c) A time-dependent problem with daily load, energy and price patterns, relatively high noise, less accurate forecasts using time series (errors grow with the horizon) and a reasonably fast storage device.

d) A time-dependent problem with daily load, energy and price patterns, relatively low noise, very accurate forecasts and a reasonably fast storage device.

e) Same as (c), but the forecast errors are stationary over the planning horizon.

Each problem variation was designed specifically to take advantage of the characteristics of each of our five policies. We tested all five policies on all five problems. In each case, we evaluated the policy by solving the problem using perfect information (this is known as a posterior bound), and then evaluating the policy as a fraction of this posterior bound. The results are shown in table 11.2, where the bold entries (in the diagonal) indicates the policy that worked best on that problem class.

Problem:	PFA	CFA-EC	VFA	LA-DET	LA-CFA
A	**0.959**	0.839	0.936	0.887	0.887
B	0.714	**0.752**	0.712	0.746	0.746
C	0.865	0.590	**0.914**	0.886	0.886
D	0.962	0.749	0.971	**0.997**	0.997
E	0.865	0.590	0.914	0.922	**0.934**

Table 11.2 Performance of each class of policy on each problem, relative to the optimal posterior solution (from Powell & Meisel (2016*b*)). Bold indicates the best performer.

The table shows that *each* of the five policies works best on one of the five problems. Of course, the problems were designed so that this was the case, but this illustrates that any of the policies can be best, even on a single problem class, just by modifying the data. For example, a deterministic lookahead works best when the forecast is quite good. A VFA-based strategy works best on problems that are very time-dependent, with a high degree of uncertainty (that is, the forecasts are poor). The hybrid CFA-based policy works best when the forecast is uncertain, but adds value.

11.10 CHOOSING THE POLICY CLASS

Given the choice of policies, the question naturally arises, how do we design a policy that is best for a particular problem? Not surprisingly, it depends on the characteristics of the problem, constraints on computation time, and the complexity of the algorithm. Below we summarize different types of problems, and provide a sample of a policy that appears to be well suited to the application, largely based on our own experiences with real problems.

11.10.1 The policy classes

We begin our discussion by reviewing the characteristics of each of our four meta-classes of policies.

Policy function approximations

A utility would like to know the value of a battery that can store electricity when prices are low and release them when prices are high. The price process is highly volatile, with a modest daily cycle. The utility needs a simple policy that is easy to implement in software. The utility chose a policy where we fix two prices, and store when prices are below the lower level and release when prices are above the higher level. This requires optimizing these two price points. A different policy might involve storing at a certain time of day, and releasing at another time of day, to capture the daily cycle.

The PFA is a natural choice because we understand the structure of the policy. It seems clear (and supporting research proves that this is the case) that a "buy low, sell high" policy is optimal. In many cases, the structure of a PFA seems apparent, but lacks any proof of optimality, and may not be optimal, but likely works quite well.

An exception to this guidance is the use of neural networks which have attracted considerable attention for controlling robots, and for playing computer games that provide an environment for collecting large numbers of observations. Neural networks can handle complex inputs such as the characteristics of a player and the state of the game. Neural networks appear to work best in low-noise environments, and where you can run large numbers of repetitions to train the typically large number of parameters that make up a neural network.

Even when the structure of the policy seems apparent, there are several problem characteristics that limit the usefulness of PFAs:

- Time dependency - It may easily be the case that the parameters of our PFA (e.g. the points at which we buy and sell electricity) are time dependent. It is relatively easy to optimize over two parameters. If there are 100 time periods, it is an entirely different matter to optimize over 200 parameters.
- State dependency - Our policy may depend on other state variables such as weather (in our energy storage attribute). In a health application, we may be able to design a PFA to determine the dosage of a medication to lower blood sugar. For example, we may be able to design a simple linear (or piecewise linear) function relating the dosage to the level of blood sugar. But the choice of drug (there are dozens) may depend on patient attributes, and we may need a different PFA for each drug (and perhaps even reflecting different patient attributes). Now our PFA is much more complex.
- Decision dimensionality - PFAs are not well suited to problems where the decision x_t is a vector. If your decision is a vector, that is a quick hint that you are going to need one of the three classes (CFAs, VFAs and DLAs) that have an imbedded optimization problem which allows us to draw on all the tools of mathematical programming.

Cost function approximation

Cost function approximations may easily be the most widely used class of policy in real applications, although as a class they have been largely ignored by the research literature.

CFAs are often used when there is a natural deterministic approximation that can be solved using standard methods. The idea is to introduce parameters that make the policy work better under uncertainty. Of course, this means that, just as with PFAs, there has to be enough structure that we can design an effective parameterization. However, rather than building a policy from scratch, we are starting with a deterministic approximation.

We first saw CFAs used very effectively in pure learning problems in chapter 7. For example, the interval estimation policy

$$X^{IE}(S^n|\theta^{IE}) = \arg\max_x \left(\bar{\mu}_x^n + \theta^{IE}\bar{\sigma}_x^n\right),$$

which trades off exploitation (by maximizing over $\bar{\mu}_x^n$ which is our estimate of how well choice x might work) and exploration (by maximizing over $\bar{\sigma}_x^n$ which is the standard deviation of our estimate $\bar{\mu}_x^n$). The weight that we put on $\bar{\sigma}_x^n$ relative to $\bar{\mu}_x^n$, given by θ^{IE}, has to be tuned.

CFAs are useful when we have an intuitive idea of how to handle uncertainty. Consider the problem of deciding on a time to leave for work for your job in a dense city. Your navigation system tells you that the trip will take 37 minutes, so you add 10 minutes to be safe. After following this strategy for a week, you arrive late one day because of an unexpected delay, so you increase your buffer to 15 minutes. This is a form of CFA which is searching for the best path, and then adding a tunable buffer to account for uncertainty.

CFAs are also well-suited to complex, high dimensional problems such as scheduling an airline. In this setting, we would solve a large, deterministic integer program to schedule planes and crews, but we have to deal with the uncertainty of flight times due to congestion and weather delays. The airline adds a buffer which may depend on both the origin and destination, but also the time of day. This buffer might be based on a dataset where the airline chooses a buffer so that the flight should be on-time θ percent of the time. The airline will then monitor network-wide on-time performance and feedback from customers to help it tune θ.

Value function approximations

Value function approximations tend to be used for problems where we need to capture the impact of a decision now on the future, *and* where this value can be captured in a well-defined function. Since policies based on VFAs are much easier to use than policies based on DLAs (but notably when we need a stochastic lookahead), the first question should be for problems in this class: Why aren't you using VFAs?

This is where you have to look at your problem and ask how complicated the value function needs to be. Note that dimensionality is not an issue. If you feel that you can reasonably approximate the future using a function that is linear or concave (if maximizing, convex if minimizing) in the state variable, these can be estimated for very high dimensions. These can often be found in large resource allocation problems.

Some examples where VFAs seem to be relatively easy to approximate are:

A blood management problem - Consider the blood management problem presented in section 8.3.2. We can use approximate dynamic programming to solve high-dimensional, spatially distributed versions of this problem using the methods we will describe in chapter 18.

Inventory problems - There are many problems where R_t is a scalar describing the inventory of product for sale, blood supplies, energy in a battery, or cash in a mutual fund.

Routing on a graph - We are at a node i and need to determine which link (i, j) to go to, where traversing a link incurs a random cost $\hat{c}_{ij}$ which is revealed after we move from i to j. We need to learn the value $\bar{v}_i$ of being at each node to make the best decision. Note that this representation is using a lookup table version of the value functions, which means the number of nodes cannot be too large.

We can easily tweak these problems to create examples where the value functions would be quite difficult to approximate:

Blood management with backlogging - Take our blood management problem from section 8.3.2, and add the simple twist that there are elective surgeries which do not have to be satisfied right away. If we do not cover a surgery now, we can perform it at a later time. This "backlogging" introduces interactions between the amount of blood on hand, and the backlogged surgeries, which makes the structure of the value function much more complicated.

Contextual inventory problems - Imagine that while managing our inventory R_t we have to consider other dynamic data. For example, if R_t is how much energy is in the battery, we might also have to keep track of the current and previous prices of energy, the temperature, and the demand for energy. This additional data is sometimes known as a "context," and it complicates the problem because the value of inventory typically does not have structural properties that we can exploit.

Routing on a dynamic graph - Imagine we face the situation of planning a path through a real network with travel times that are constantly being updated. Although often overlooked, the state variable for this problem is a combination of the node where a vehicle has to make a decision, and the current estimates of the link times *for every link in the network!*.

There are problems where policies based on value function approximations represent an amazing breakthrough, but we imagine that they are a very small percentage of real sequential decision problems (which are ubiquitous).

Direct lookahead policies

There are many problems which just naturally seem to require that we plan over a horizon to make a decision now. An easy example is a navigation system that plans a path all the way to the destination to determine whether to turn right or left at the next intersection. This problem could never be solved with a PFA or CFA. One can argue that it can be solved with a VFA because deterministic shortest path problems are, in fact, dynamic programs that are solved using value functions, but this is only after we have translated the problem to a deterministic approximation (that is, a DLA), ignoring rolling forecasts.

There are three important strategies in the DLA class that are quite practical for many applications:

Deterministic lookahead - Sometimes known as model predictive control (MPC) or a rolling/receding horizon procedure, a deterministic lookahead is often the first policy that many will try when faced with a problem which needs a lookahead policy. There is not a simple formula that determines this, but readers should think about their problem and ask to what extent downstream decisions might affect a decision that you need to make now. Also important when choosing between a DLA and VFA is to

what extent information is treated as a latent variable in the VFA. For example, it is very common for forecasts to be modeled as latent variables when using VFA-based policies, which means the VFAs have to be recomputed each time the forecasts are updated. By contrast, deterministic DLAs have the forecast built right into the model, which is often (but not always) relatively easy to solve.

We have found that some quantities are easier to approximate deterministically than others. For example, we would never obtain an approximation of a buy-low, sell-high type of policy if we model uncertain prices with a deterministic forecast. On the other hand, we seem to be quite comfortable planning the best path over a network using point estimates of travel times.

As a general pattern, as uncertainty increases, VFAs tend to work better, since they provide a natural mechanism for smoothing over variations. Higher levels of uncertainty also tend to make value functions smoother and easier to approximate.

Deterministic lookaheads can often be good approximations even in the presence of uncertainty. For example, it works quite well in planning paths to a destination even though travel times over each leg of the network are random.

Sampled lookaheads - When we need to handle uncertainty in the future, we are going to return to our approximate, stochastic lookahead policy which we write as

$$X_t^{LA-Stoch}(S_t) = \arg\max_{x_t} \Bigg(C(S_t, x_t) + \tilde{E}\left\{ \max_{\tilde{\pi}} \tilde{E}\left\{ \sum_{t'=t+1}^{T} C(\tilde{S}_{tt'}, \tilde{X}_t^{\tilde{\pi}}(\tilde{S}_{tt'}|\tilde{\theta}_t))|\tilde{S}_{t,t+1} \right\} |S_t, x_t \right\} \Bigg). \quad (11.42)$$

While equation (11.42) can look frightening, we are going to break it down in chapter 19. In a nutshell, each $\tilde{E}$ is approximated by simulating the underlying sequence $\widetilde{W}_{t,t+1}$ (for the first $\tilde{E}$) and then $\widetilde{W}_{t,t+2}, \widetilde{W}_{t,t+3}, \ldots, \widetilde{W}_{t,t+H}$ (for the second $\tilde{E}$). The challenge is designing the lookahead policy $\tilde{X}^{\tilde{\pi}}(\tilde{S}_{tt'})$. While of course we want the best possible, in a lookahead model we can get away with less than the best policy, focusing more attention to computational complexity.

Parameterized lookaheads - A strategy that is widely used in practice, but largely ignored by the research community, is the idea of using a deterministic lookahead, but then introduce parameters that modify the deterministic model so that it works better under uncertainty. For example, imagine that we want to find the best path over a network with uncertain link times. Instead of using the estimate of the average link time, we might use the $80th$ percentile. Now turn this percentile into a tunable parameter θ and simulating use shortest paths based on these travel times.

11.10.2 Policy complexity-computational tradeoffs

There is a simple tradeoff when choosing policies. Simply put: the more work you put into computing your policy, the less work you have to put into designing and tuning it.

For example, PFAs are the simplest functions, but they only work well for simple problems. Inventory problems are a nice example. Standard inventory policies are characterized by a lower inventory θ^{min} that triggers an inventory order, and an order-up-to amount θ^{max},

State variables	Future information	Future decisions
Cargo ships will arrive in 6 and 20 weeks	A ship due in 40 days may be delayed 0 to 7 days	We can send rush order via air freight
A storm will hit the port creating a 1 week delay	Demand for produce may shift up by 15 percent	We can raise prices
A surge in demand will occur in 2 weeks	Forecasted transportation capacity cannot meet the surge	Outside transportation capacity has to be arranged
A commodity price just jumped 20 percent	Commodity shortages may arise	We can change suppliers

Table 11.3 Illustration of complicating state variables, future information and future decisions for our decision problem.

creating a tunable parameter vector $\theta = (\theta^{min}, \theta^{max})$. This sounds so deceptively simple that the inventory literature has not progressed past this elementary policy in 60 years.

In fact, the opening section of the book started with an inventory problem of goods crossing the Pacific to a warehouse in the Southeastern U.S. (see figure 1.1) which introduces a sequence of complications. We have created a sample of these complications in table 11.3 where we have listed three types of issues that our policy would have to consider: additional state variables, future information, and future decisions.

The complicating state variables mean that our state is no longer our inventory R_t, but a host of other information such as the timing of previously ordered inventory (the ships arriving in 6 and 20 weeks), the storm about to hit the port and the change in commodity prices. We can roll all this into our state S_t, but how does this change our inventory policy?

Now our order-up-to parameter vector θ becomes a function $\theta(S_t)$, but what does this function look like? Most likely this will involve additional experimentation and more parameters that have to be tuned. The PFA may be simple, but the state-dependent parameter vector $\theta(S_t)$ represents a major complication. In addition, the challenge of nonstationary behavior typically means that $\theta(S_t)$ becomes $\theta_t(S_t)$, which means θ itself (or the function) is now time dependent.

Additional complications arise in the information that *might* arrive in the future, and then the decisions we *might* make in response. However, the fundamental structure of an order-up-to policy is built around a fairly simple model that can not handle the richness of a hybrid decision structure that adapts to new information in different ways.

The other classes of policies are better suited at handling these complexities. VFAs are better suited at handling time dependencies and high levels of uncertainties, while DLAs (which are often parameterized), which require solving an approximate lookahead, remove much of the guesswork of how a policy should behave by building the complicating issues directly into the model. The price is additional computation (and possibly quite a bit more).

At the risk of oversimplifying this issue, figure 11.3 depicts the tradeoff between the complexity of creating a policy and the cost of computing it. We have divided direct lookaheads between deterministic and stochastic lookaheads, since they are dramatically different. The point is to recognize that the complexities of building a policy and computing it are important issues that have to be considered when designing a policy.

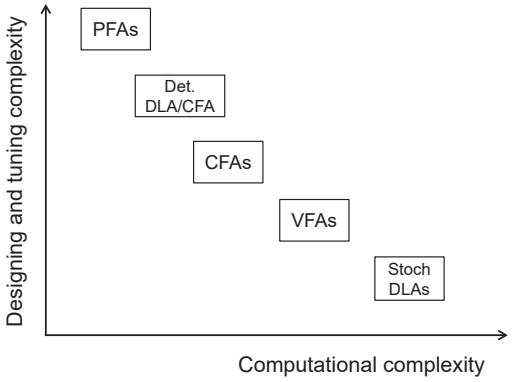


Figure 11.3 Illustration of the tradeoff between the complexity of creating a policy, and the cost of computing a policy, for each of the major policy classes.

11.10.3 Screening questions

The following screening questions may help to assess which policy class may be appropriate. Keep in mind that the choice of policy is heavily dependent on the context of specific applications which you are working on.

1) Does the problem have structure that suggests a simple and natural decision rule? If there is an "obvious" policy (e.g. replenish inventory when it gets too low), then more sophisticated algorithms based on value function approximations are likely to struggle. Exploiting structure always helps, but always remember: simple rules imply tunable parameters, and tuning can be hard.

2) Would a greedy (that is, myopic) policy work reasonably well? This would open the door to solving relatively easy optimization problems for high-dimensional resource allocation problems (such as assigning resources to tasks).

3) Is the problem fairly stationary, or highly nonstationary? Nonstationary problems (e.g. responding to hourly demand or daily water levels) mean that you need a policy that depends on time. Rolling horizon problems can work well if the level of uncertainty is low relative to the predictable variability. It is hard to produce policy function approximations where the parameters vary by time period.

4) Do you have a strong sense that decisions that you might make in the future will affect what you are going to do now? An easy example is a vehicle navigation system that plans the path all the way to the destination, but planning investments to meet major obligations (college, retirement) is another example. If this is the case, we then need to ask questions about uncertainty:

   a) Do you have a well-defined goal you have to reach by a point in time?

   b) Could a deterministic approximation of the future be a reasonable starting point? Since deterministic lookahead policies are so popular (there is an entire field of optimal control, called model predictive control, dedicated to this approach), we have to think carefully about "why not a deterministic lookahead." However,

there are many problems where deterministic lookaheads would not be effective. Some examples are:

- Asset selling problems with stochastic selling prices - Optimal policies depend very much on policies that recognize price variations and exploit them (e.g. sell when price goes above some point).
- Managing a single resource serving discrete demands (see the nomadic trucker example in section 2.3.4).
- There are actually dynamic problems where a deterministic lookahead model is too large to be solved quickly enough in a dynamic environment.

c) How much uncertainty is in the future? Value function approximations are especially valuable when uncertainty is high, and can make the value of being in a state easier to approximate.

5) Does the value of the most important state variables (in particular physical states or states that are being directly controlled) appear to have natural structure that can be exploited?

A guiding principle when working with the four classes of policies is to start with the simplest policies and then work up.

Table 11.4 lists each of the scenarios above with a suggested starting strategy. Given the massive diversity in problem classes, it is exceptionally difficult to provide precise advice, but we emphasize: these are only intended as starting suggestions.

Unless you are pursuing an algorithm as an intellectual exercise, it is best to focus on your problem and choose the method that is best suited to the application. For more complex problems, be prepared to use a hybrid strategy. For example, rolling horizon procedures may be combined with adjustments that depend on tunable parameters (a form

Scenario	Recommended strategy:
1)	Clear choice for a PFA.
2)	Likely choice for a CFA since this likely requires an imbedded optimization problem. Look for parameterizations to improve performance.
3)	A deterministic lookahead (with imbedded forecast) can turn a nonstationary problem into a stationary one.
4)	At this point you are headed down the class of direct lookahead policies; just have to figure out which one.
4a)	If you can live with a deterministic lookahead, then this is your first step. If you need to reach a specific target by a specific time under uncertainty, then you are looking at a technically challenging policy.
4b)	This suggests a deterministic direct lookahead is your natural starting point.
4c)	If you have significant uncertainty, then deterministic lookaheads start to struggle, and a value function starts to become more attractive.
5)	If the value of the state variables that are directly being controlled (even some relatively simple systems can have auxiliary variables that evolve exogenously), then try designing an architecture for a value function approximation. At this point, you need consider the updating mechanisms that are described in chapters 15-18.

Table 11.4 Suggested starting strategies for the different scenarios in the text.

of policy function approximation). You might use a lookahead policy using a decision tree combined with a simple value function approximation to help reduce the size of the tree.

CFAs incorporate more structure in the optimization problem, which makes tuning the coefficients easier. A pure CFA does not attempt to approximate the future, while VFA-based policies are approximating the future, inside an optimization problem (there is an imbedded $\arg\max_x$ within the VFA policy), which tends to further simplify coming up with both the architecture of the VFA, as well as the tuning.

Keep in mind that any approximation can be compensated with tunable parameters. This is how PFAs and (myopic) CFAs (such as UCB policies) can be effective. However, approximate DLAs can also be compensated using well designed parameterizations, as we are going to demonstrate in chapter 13. Also, tuning a parameterized DLA is simplified because it already has a considerable amount of the problem structure built into policy, as we are going to demonstrate in 13.

But ultimately, it depends on the characteristics of your particular application.

11.11 POLICY EVALUATION

The choice of the best policy class, and in particular any tuning within a policy class, requires that we perform policy evaluation.

We first have to decide: are we maximizing cumulative reward, as would typically happen in an online setting? In this case we would use

$$\begin{aligned}\max_{\pi} F^{\pi} &= \mathbb{E}\left\{\sum_{t=0}^{T} C(S_t, X^{\pi}(S_t), W_{t+1})|S_0\right\} \\ &= \mathbb{E}_{S_0}\mathbb{E}_{W_1,\ldots,W_T|S_0}\left\{\sum_{t=0}^{T} C(S_t, X^{\pi}(S_t))|S_0\right\}.\end{aligned}$$

We then simulate F^{π} using

$$F^{\pi}(\theta|\omega) = \sum_{t=0}^{T-1} C(S_t(\omega), X^{\pi}(S_t(\omega)|\theta)).$$

Finally we average over sample paths to obtain

$$\bar{F}^{\pi}(\theta) = \frac{1}{K}\sum_{k=1}^{K} F^{\pi}(\theta|\omega^k).$$

Otherwise, we are optimizing a final design, which for state-dependent problems, means we are evaluating our policy using

$$\begin{aligned}\max_{\pi^{lrn}} F^{\pi^{lrn}} &= \mathbb{E}\{C(S, X^{\pi^{imp}}(S|\theta^{imp}), \widehat{W})|S^0\} \\ &= \mathbb{E}_{S^0}\mathbb{E}^{\pi^{imp}}_{(W_t^n)_{t=0}^T)_{n=0}^N|S^0}\left(\mathbb{E}^{\pi^{imp}}_{(\widehat{W}_t)_{t=0}^T|S^0}\frac{1}{T}\sum_{t=0}^{T-1} C(S_t, X^{\pi^{imp}}(S_t|\theta^{imp}), \widehat{W}_{t+1})\right).\end{aligned}$$

We then simulate F^{π} using

$$F^{\pi}(\theta^{lrn}|\omega,\psi) = \frac{1}{T}\sum_{t=0}^{T-1} C(S_t(\omega), X^{\pi^{imp}}(S_t(\omega)|\theta^{imp}), \widehat{W}_{t+1}(\psi)).$$

Finally we average over sample paths to obtain

$$\bar{F}^{\pi}(\theta^{lrn}) = \frac{1}{K}\frac{1}{L}\sum_{k=1}^{K}\sum_{\ell=1}^{L} F^{\pi}(\theta^{lrn}|\omega^k, \psi^{\ell}).$$

An important part of the evaluation is designing the observations of the testing samples represented by $\widehat{W}$.

A separate choice is the handling of risk. We use an expectation operator $\mathbb{E}$ as our default metric for averaging across outcomes, but it is entirely possible that risk is an important issue. We might be interested in worst-case performance, 10^{th}-percentile, or one of the VaR or CVaR risk measures discussed in section 9.8.4 are relevant.

11.12 PARAMETER TUNING

Parameter tuning in policy search is its own stochastic optimization problem to find a policy (or algorithm) to solve a stochastic optimization problem which we can write as

$$\max_{\theta} F^{\pi}(\theta). \tag{11.43}$$

Since $F^{\pi}(\theta)$ involves an expectation we cannot compute we typically are solving

$$\max_{\theta} \bar{F}^{\pi}(\theta). \tag{11.44}$$

Independent of which class of problem produces our function $F^{\pi}(\theta)$ (or $\bar{F}^{\pi}(\theta)$), we need to find a (possibly vector-valued) parameter θ that controls our implementation policy (or how we find our implementation policy).

There are two broad strategies for performing parameter tuning: derivative-based stochastic search (which we covered in chapter 5) and derivative-free stochastic search (covered in chapter 7). Remember that we can use numerical derivatives for problems where gradients are not directly available (which is most of the time). The SPSA algorithm (see section 5.4.4) is well suited for optimizing vector-valued parameters θ even when derivatives are not available.

The process of parameter tuning will need to consider the following issues:

Simulators vs. field experiments - It has been our experience that the vast majority of formal parameter tuning is done using simulators, but building a well-calibrated simulator can be a major project. There are many sequential decision problems that need to be solved, but which do not justify the resources required to build a simulator. If this is the case, the only alternative is to use online learning in the field, which eliminates the possibility of using any derivative-based algorithm. The techniques of chapter 7, using a cumulative-reward objective (since you have to learn while doing).

Tunable parameters - Choosing the best policy is going to require balancing computational complexity against the simplicity of parameterized policies. The simpler policies in the PFA and CFA classes will look appealing because of their simplicity and easy of development, but as you gain experience in this area, you will start to appreciate the line: "The price of simplicity is tunable parameters, and tuning is hard!" The lookahead policies typically have a much lower burden of parameter tuning (and when

there are parameters, they are easier to tune), but you trade off the computational cost of executing these policies in the field.

Latent variables - Further complicating the process of parameter tuning is the presence of "latent variables." Latent variables are, by definition, hidden, which means that if they change, their effect is not being modeled explicitly. A latent variable can be as simple as the starting point of an algorithm. If you tune the parameters of a stepsize rule for a particular starting point, the resulting stepsize rule may easily fail with starting points that are much closer to or farther from the optimal solution. Latent variables can also be the noise in an experiment, or problem features that affect the shape of the response surface.

Expensive experiments - There are many settings where experiments are time consuming (and possibly expensive). Any experiments in the field face the problem that it takes a day to observe a day. However, there are problems that require expensive computer simulations spanning hours to days for a single observation. Laboratory experiments are typically much worse. As of this writing, the research on parameter tuning with small budgets is quite limited. The key in such problem settings is exploiting as much structure and domain knowledge as possible.

Throughout the parameter tuning process, remember that parameter tuning is a sequential decision problem to solve a sequential decision problem. To get a good solution to your real application, you have to do a good job with the parameter search. We recommend testing your search procedure on some benchmark application that allows you to get an accurate measure of how well the procedure is working. Of course, you want to design a benchmark that matches the general behavior of your real application.

Just as a weak algorithm for a deterministic optimization problem can produce a poor solution, a weak search algorithm ("learning policy") can produce a poor implementation policy. In fact, the results can be quite poor. Just because you run an algorithm many iterations does not mean that you have produced a high quality (or even good) solution. The best way to protect yourself is to design competing solution approaches (perhaps using two or more classes of policies, but this even applies within a class) and choose the one that works best.

11.12.1 The soft issues

If the number of classes being tested is small, a reasonable strategy is to analyze each of the policy classes and choose the best one. Of course, we can do better, since this is basically a search over discrete choices.

Rather than evaluate each policy class in depth (which is impractical), we can do a partial evaluation, just as we would examine an unknown function. This introduces the issue of having to optimize over a set of parameters in order to evaluate a particular search policy/algorithm. If this is easy, then finding the best search policy/algorithm may not be as critical. However, imagine finding the best search policy for a problem where derivatives are not available, and function evaluations take several hours (or a day). Choosing the policy is not as obvious as it may seem, given our focus on finding optimal policies. Unlike deterministic optimization, where we want the *best* solution x (lowest cost, highest profit, ...), how well the policy $X^\pi(S_t|\theta)$ performs is only one of a number of factors to be considered.

There are parallels with machine learning where we want the value of θ so that our model $f(x|\theta)$ produces the best fit to the data (the training dataset). However, choosing the best model $f(x|\theta)$, which requires searching over functions $f \in \mathcal{F}$, is more complicated. The goal is to work well in the field, and while producing good estimates (or predictions) is always important, issues such as transparency and robustness are also important.

Choosing the best policy depends on the context, but a list of important issues that can and will be important in the final choice include:

- Solution quality - Of course we would like solutions that perform as well as possible, especially in higher volume transactions with clear economic consequences.
- Computational tractability - A representative from Google once made a statement that they wanted the best policy for choosing what ads to display, but it could not take more than 50 milliseconds. A major grid operator has four hours to determine their plan for generation for tomorrow, but they are being asked to implement stochastic lookaheads (which we address in chapter 19) that require significantly more computational effort.
- Robustness - Is the procedure consistently reliable, across a wide range of conditions?
- Methodological complexity - If the method is captured in a black box package, then we only care if the package works, and how well. But there are very few general purpose packages, which means companies (or their consultants) have to develop the logic on their own. A company (more precisely the team doing the work) has to feel confident that the method can be implemented correctly, with good results, on time, and on budget.
- Transparency/diagnosability - We may need to understand *why* a decision is made. If an automated system turns down a loan application from a minority applicant, laws may require that this be documented. However, we may also wonder why a driver is moving a long empty move to pick up a load: Did the load have to be moved? Could it be moved later? Since data may not be perfect, it may be necessary to understand what data is having an impact on the decision. If we do not like a decision, can we trace the reasons behind the recommendation so that we can either understand it, or fix it?
- Data requirements - We need to understand what data is required, and how reliable it is.

11.12.2 Searching across policy classes

The previous section focused on tuning the parameters of a particular policy class. What about searching across policy classes? We need to remember that the four "classes" of policies are really meta-classes; picking a class such as PFA or VFA still involves a lot of work identifying the best functional approximation (for the policy or value function), and then doing all the work of tuning or fitting these approximations. It is not unusual to spend several months developing a particular policy. Doing this for each of the policy classes is generally going to be impractical.

This is where it will be necessary to think of the issues raised in this chapter. Soft issues may dominate the choice of policy class. How much time do you have to develop and test a policy? How important is computational complexity, or transparency? There is nothing wrong with letting these dimensions steer the choice of policy class. For the reader who is

using this book to solve a specific problem (rather than gaining general knowledge of the field), our hope is that the discussion in this chapter might guide them to the chapter that will best fit the needs of your problem.

11.13 BIBLIOGRAPHIC NOTES

Section 11.2 - The identification that there are specific classes of policies was first proposed in Powell (2011)[Chapter 6], but this discussion failed to identify cost function approximations as a specific class. The four classes of policies as they are identified in this book were first formalized in Powell (2014*b*). Powell (2016*a*) divided the four classes into the two core strategies: "policy search policies," and "lookahead policies." Finally, Powell (2019*a*) introduced the concepts of state-independent problems (pure learning problems) and state-dependent problems, along with final-reward and cumulative-reward objectives.

This chapter gives a quick overview of all four classes of policies, but this is just to lay the foundation for the rest of the book. Each of these four classes have been studied in depth, and will be covered in chapters 12-19. Please look at the bibliographic notes in these chapters for more complete summaries of references.

Section 11.9 - This work was taken from Powell & Meisel (2016*b*).

EXERCISES

Review questions

11.1 What is a policy?

11.2 What are the two strategies for designing policies? What distinguishes them?

11.3 Each of the two strategies consists of two classes of policies. Name them, and describe the distinguishing characteristics of each of the four classes that separates them from the other three.

11.4 For each of the four classes of policies, describe the characteristic(s) that are most difficult about that class.

11.5 What is the central message of the energy storage problem described in section 11.9?

11.6 What is meant by the "policy-within-a-policy"?

11.7 Describe what is meant by a randomized policy? Give an examples of randomized policies for a) continuous decisions and b) discrete decisions.

Modeling questions

11.8 What is the difference between a stationary policy, a deterministic nonstationary policy, and an adaptive policy?

11.9 Below is a list of problems with a proposed method for making decisions. Classify each method based on the four classes of policies (you may decide that a method is a hybrid of more than one class).

a) You use Google maps to find the best path to your destination.

b) You are managing a shuttle service between the mainland and a small resort island. You decide to dispatch the shuttle as soon as you reach a minimum number of people, or when the wait time of the first person to board exceeds a particular amount.

c) An airline optimizes its schedule over a month using schedule slack to protect against potential delays.

d) Upper confidence bounding policies for performing sequential learning (these were introduced in chapter 7).

e) A computer program for playing chess using a point system to evaluate the value of each piece that has not yet been captured. Assume it chooses the move that leaves it with the highest number of points after one move.

f) Imagine an improved computer program that enumerates all possible chess moves after three moves, and then applies its point system.

g) Thompson sampling for sequential learning (also introduced in chapter 7).

11.10 You are the owner of a racing team, and you have to decide whether to keep going with your current driver or to stop and consider a new driver. The decision after each race is

to stay with your driver or stop (and switch). The only outcome you care about is whether your driver won or not.

a) Formulate the problem as a decision tree over three races (we index these races as 0, 1 and 2).

b) In equation (11.23), we write our optimal policy as

$$X_t^*(S_t) = \arg\max_{x_t} \left(C(S_t, x_t) + \mathbb{E} \left\{ \max_{\pi} \mathbb{E} \left\{ \sum_{t'=t+1}^{T} C(S_{t'}, X_{t'}^{\pi}(S_{t'})) \middle| S_{t+1} \right\} | S_t, x_t \right\} \right). \quad (11.45)$$

Letting $t = 0$ where we face one of two actions (stay with current driver or replace), fully enumerate all the policies we may consider for $t = 1, 2$.

c) The outer expectation $\mathbb{E}$ in (11.45) is over which random variable(s)?

d) The inner expectation $\mathbb{E}$ in (11.45) is over which random variable(s)?

Problem solving questions

11.11 Following is a list of how decisions are made in specific situations. For each, classify the decision function in terms of which of the four fundamental classes of policies are being used. If a policy function approximation or value function approximation is used, identify which functional class is being used:

- If the temperature is below 40 degrees F when I wake up, I put on a winter coat. If it is above 40 but less than 55, I will wear a light jacket. Above 55, I do not wear any jacket.
- When I get in my car, I use the navigation system to compute the path I should use to get to my destination.
- To determine which coal plants, natural gas plants and nuclear power plants to use tomorrow, a grid operator solves an integer program that plans over the next 24 hours which generators should be turned on or off, and when. This plan is then used to notify the plants who will be in operation tomorrow.
- A chess player makes a move based on her prior experience of the probability of winning from a particular board position.
- A stock broker is watching a stock rise from $22 per share up to $36 per share. After hitting $36, the broker decides to hold on to the stock for a few more days because of the feeling that the stock might still go up.

11.12 Repeat exercise 11.11 for the following decision situations:

- A utility has to plan water flows from one reservoir to the next, while ensuring that a host of legal restrictions will be satisfied. The problem can be formulated as a linear program which enforces these constraints. The utility uses a forecast of rainfalls over the next 12 months to determine what it should do right now.
- The utility now decides to capture uncertainties in the rainfall by modeling 20 different scenarios of what the rainfall might be on a month-by-month basis over the next year.

- A mutual fund has to decide how much cash to keep on hand. The mutual fund uses the rule of keeping enough cash to cover total redemptions over the last 5 days.
- A company is planning sales of TVs over the Christmas season. It produces a projection of the demand on a week by week basis, but does not want to end the season with zero inventories, so the company adds a function that provides positive value for up to 20 TVs.
- A wind farm has to make commitments of how much energy it can provide tomorrow. The wind farm creates a forecast, including an estimate of the expected amount of wind and the standard deviation of the error. The operator then makes an energy commitment so that there is an 80 percent probability that he will be able to make the commitment.

11.13 Consider two policies:

$$X^{\pi^A}(S_t|\theta) = \arg\max_{x_t}\left(C(S_t,x_t) + \sum_{f\in\mathcal{F}}\theta_f\phi_f(S_t)\right), \tag{11.46}$$

and

$$X^{\pi^B}(S_t|\theta) = \arg\max_{x_t}\left(C(S_t,x_t) + \sum_{f\in\mathcal{F}}\theta_f\phi_f(S_t)\right). \tag{11.47}$$

In the case of the policy π^A in equation (11.46), we search for the parameter vector θ by solving

$$\max_{\theta}\mathbb{E}\sum_{t=0}^{T}C(S_t,X^{\pi^A}(S_t|\theta)). \tag{11.48}$$

In the case of policy π^B, we wish to find θ so that

$$\sum_{f\in\mathcal{F}}\theta_f\phi_f(S_t) \approx \mathbb{E}\sum_{t'=t}^{T}C(S_t,X^{\pi^B}(S_t|\theta)). \tag{11.49}$$

a) (5 points) Classify policies π^A and π^B among the four classes of policies.

b) (5 points) Can we expect that the value θ^A that optimizes (11.48) would be approximately equal to the value θ^B that solves equation (11.49)?

c) Assuming that we can solve the policy search problem in (11.48) to optimality, can we make a statement about which of the two policies might be better? Explain.

11.14 Earlier we considered the problem of assigning a resource i to a task j. If the task is not covered at time t, we hold it in the hopes that we can complete it in the future. We would like to give tasks that have been delayed more higher priority, so instead of just just maximizing the contribution c_{ij}, we add in a bonus that increases with how long the task has been delayed, giving us the modified contribution

$$c^{\pi}_{tij}(\theta) = c_{ij} + \theta_0 e^{-\theta_1(\tau_j - t)}.$$

Now imagine using this contribution function, but optimizing over a time horizon T using forecasts of tasks that might arrive in the future.

a) Write out the objective function for optimizing θ offline in a simulator.

b) Would solving this problem, using $c^{\pi}_{tij}(\theta)$ as the contribution for covering task j using resource i at time t, give you the behavior that you want?

Diary problem

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

11.15 List all the decisions that arise in the context of your diary problem (there may be only one, but if your problem is sufficiently rich, you can probably find several). Suggest the class of policy you think is most promising for each type of decision. If possible, try to identify a second choice, and discuss why you feel that the first choice is better.

11.16 Discuss the soft issues (section 11.13) that you anticipate would be relevant to at least one of the decisions in your diary problem?