REINFORCEMENT LEARNING AND STOCHASTIC OPTIMIZATION

A unified framework for sequential decisions

Warren B. Powell

August 22, 2021

WILEY-INTERSCIENCE

A JOHN WILEY & SONS, INC., PUBLICATION

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 7

DERIVATIVE-FREE STOCHASTIC SEARCH

There are many settings where we wish to solve

$$\max_{x \in \mathcal{X}} \mathbb{E}\{F(x, W)|S^0\}, \tag{7.1}$$

which is the same problem that we introduced in the beginning of chapter 5. When we are using derivative-free stochastic search, we assume that we can choose a point x^n according to some policy that uses a belief about the function that we can represent by $\bar{F}^n(x) \approx \mathbb{E}F(x, W)$ (as we show below, there is more to the belief than a simple estimate of the function). Then, we observe the performance $\hat{F}^{n+1} = F(x^n, W^{n+1})$. Random outcomes can be the response of a patient to a drug, the number of ad-clicks from displaying a particular ad, the strength of a material from a mixture of inputs and how the material is prepared, or the time required to complete a path over a network. After we run our experiment, we use the observed performance $\hat{F}^{n+1}$ to obtain an updated belief about the function, $\bar{F}^{n+1}(x)$.

We may use derivative-free stochastic search because we do not have access to the derivative (or gradient) $\nabla F(x, W)$, or even a numerical approximation of the derivative. The most obvious examples arise when x is a member of a discrete set $\mathcal{X} = \{x_1, \ldots, x_M\}$, such as a set of drugs or materials, or perhaps different choices of websites. In addition, x may be continuous, and yet we cannot even approximate a derivative. For example, we may want to test a drug dosage on a patient, but we can only do this by trying different dosages and observing the patient for a month.

There may also be problems which can be solved using a stochastic gradient algorithm (possibly using numerical derivatives). It is not clear that a gradient-based solution is

necessarily better. We suspect that if stochastic gradients can be calculated directly (without using numerical derivatives), that this is likely going to be the best approach for high-dimensional problems (fitting neural networks are a good example). But there are going to be problems where both methods may apply, and it simply will not be obvious which is the best approach.

We are going to approach our problem by designing a policy (or algorithm) $X^\pi(S^n)$ that chooses $x^n = X^\pi(S^n)$ given what we know about $\mathbb{E}\{F(x,W)|S^0\}$ as captured by our approximation

$$\bar{F}^n \approx \mathbb{E}\{F(x,W)|S^0\}$$

For example, if we are using a Bayesian belief for discrete $x \in \mathcal{X} = \{x_1, \ldots, x_M\}$, our belief B^n would consist of a set of estimates $\bar{\mu}_x^n$ and precisions β_x^n for each $x \in \mathcal{X}$. Our belief state is then $B^n = (\bar{\mu}_x^n, \beta_x^n)_{x\in\mathcal{X}}$ which is updated using, for $x = x^n$,

$$\begin{aligned}
\bar{\mu}_x^{n+1} &= \frac{\beta_x^n \bar{\mu}_x^n + \beta^W W^{n+1}}{\beta_x^n + \beta^W}, \\
\beta_x^{n+1} &= \beta_x^n + \beta^W.
\end{aligned}$$

We first saw these equations in chapter 3. Alternatively, we might use a linear model $f(x|\theta)$ which we would write

$$f(x|\bar{\theta}^n) = \bar{\theta}_0^n + \bar{\theta}_1^n \phi_1(x) + \bar{\theta}_2^n \phi_2(x) + \bar{\theta}_2^n \phi_2(x) + \ldots,$$

where $\phi_f(x)$ is a feature drawn from the input x, which could include data from a website, a movie, or a patient (or patient type). The coefficient vector $\bar{\theta}^n$ would be updated using the equations for recursive least squares (see section 3.8) where the belief state B^n consists of the estimates of the coefficients $\bar{\theta}^n$ and a matrix M^n.

After choosing $x^n = X^\pi(S^n)$, then observing a response $\hat{F}^{n+1} = F(x^n, W^{n+1})$, we update our approximation to obtain $\bar{F}^{n+1}$ which we capture in our belief state S^{n+1} using the methods we presented in chapter 3. We represent the updating of beliefs using

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

This could be done using any of the updating methods described in chapter 3. This process produces a sequence of states, decisions, and information that we will typically write as

$$(S^0, x^0 = X^\pi(S^0), W^1, S^1, x^1 = X^\pi(S^1), W^2, \ldots, S^n, x^n = X^\pi(S^n), W^{n+1}, \ldots).$$

In real applications, we have to stop at some finite N. This changes our optimization problem from the asymptotic formulation in (7.1) to the problem (which we now state using the expanded form):

$$\max_\pi \mathbb{E}_{S^0}\mathbb{E}_{W^1,\ldots,W^N|S^0}\mathbb{E}_{\widehat{W}|S^0}\{F(x^{\pi,N}, \widehat{W})|S^0\}. \tag{7.2}$$

where $x^{\pi,N}$ depends on the sequence $W^1, \ldots, W^N$.

This is the final-reward formulation that we discussed in chapter 4. We can also consider a cumulative reward objective given by

$$\max_\pi \mathbb{E}_{S^0}\mathbb{E}_{W^1,\ldots,W^N|S^0}\left\{\sum_{n=0}^{N-1} F(X^\pi(S^n), W^{n+1})|S^0\right\}. \tag{7.3}$$

For example, we might use (7.2) when we are running laboratory experiments to design a new solar panel, or running computer simulations of a manufacturing process that produces the strongest material. By contrast, we would use (7.3) if we want to find the price that maximizes the revenue from selling a product on the internet, since we have to maximize revenues over time while we are experimenting. We note here the importance of using the expanded form for expectations when comparing (7.2) and (7.3).

An entire book could be written on derivative-free stochastic search. In fact, entire books and monographs have been written on specific versions of the problem, as well as specific classes of solution strategies. This chapter is going to be a brief tour of this rich field.

Our goal will be to provide a unified view that covers not only a range of different formulations (such as final reward and cumulative reward), but also the different classes of policies that we can use. This will be the first chapter where we do a full pass over all four classes of policies that we first introduced in chapter 1. We will now see them all put to work in the context of pure learning problems. We note that in the research literature, each of the four classes of policies are drawn from completely different fields. This is the first time that all four are illustrated at the same time.

7.1 OVERVIEW OF DERIVATIVE-FREE STOCHASTIC SEARCH

There are a number of dimensions to the rich problem class known as derivative-free stochastic search. This section is designed as an introduction to this challenging field.

7.1.1 Applications and time scales

Examples of applications that arise frequently include:

- Computer simulations - We may have a simulator of a manufacturing system or logistics network that models inventories for a global supply chain. The simulation may take anywhere from several seconds to several days to run. In fact, we can put in this category any setting that involves the computer to evaluate a complex function.

- Internet applications - We might want to find the ad that produces the most ad-clicks, or the features of a website that produce the best response.

- Transportation - Choosing the best path over a network - After taking a new position and renting a new apartment, you use the internet to identify a set of K paths - many overlapping, but covering modes such as walking, transit, cycling, Uber, and mixtures of these. Each day you get to try a different path x to try to learn the time required μ_x to traverse path x.

- Sports - Identifying the best team of basketball players - A coach has 15 players on a basketball team, and has to choose a subset of five for his starting lineup. The players vary in terms of shooting, rebounding and defensive skills.

- Laboratory experiments - We may be trying to find the catalyst that produces a material of the highest strength. This may also depend on other experimental choices such as the temperature at which a material is baked, or the amount of time it is exposed to the catalyst in a bath.

- Medical decision making - A physician may wish to try different diabetes medications on a patient, where it may take several weeks to know how a patient is responding to a drug.
- Field experiments - We may test different products in a market, which can take a month or more to evaluate the product. Alternatively, we may experiment with different prices for the product, where we may wait several weeks to assess the market response. Finally, a university may admit students from a high-school to learn how many accept the offer of admission; the university cannot use this information until the next year.
- Policy search - We have to decide when to store energy from a solar array, when to buy from or sell to the grid, and how to manage storage to meet the time varying loads of a building. The rules may depend on the price of energy from the grid, the availability of energy from the solar array, and the demand for energy in the building. Policy search is typically performed in a simulator, but may also be done in the field.

These examples bring out the range of time scales that can arise in derivative-free learning:

- Fractions of a second to seconds - Running simple computer simulations, or assessing the response to posting a popular news article.
- Minutes - Running more expensive computer simulations, testing the effect of temperature on the toxicity of a drug.
- Hour - Assessing the effect of bids for internet ads.
- Hours to days - Running expensive computer simulations, assessing the effect of a drug on reducing fevers, evaluating the effect of a catalyst on materials strength.
- Weeks - Test marketing new products and testing prices.
- Year - Evaluating the performance of people hired from a particular university, observing matriculation of seniors from a high school.

7.1.2 The communities of derivative-free stochastic search

Derivative-free search arises in so many settings that the literature has evolved in a number of communities. It helps to understand the diversity of perspectives.

Statistics The earliest paper on derivative-free stochastic search appeared in 1951, which interestingly appeared in the same year as the original paper for derivative-based stochastic search.

Applied probability The 1950's saw the first papers on "one-armed" and "two-armed" bandits laying the foundation for the multiarmed bandit literature that has emerged as one of the most visible communities in this field (see below).

Simulation In the 1970's the simulation community was challenged with the problem of designing manufacturing systems. Simulation models were slow, and the challenge was finding the best configuration given limited computing resources. This work became known as "simulation optimization."

Figure 7.1 A set of slot machines.

Geosciences Out in the field, geoscientists were searching for oil and faced the problem of deciding where to dig test wells, introducing the dimension of evaluating surfaces that were continuous but otherwise poorly structured.

Operations research Early work in operations research on derivative-free search focused more on optimizing complex deterministic functions. The OR community provided a home for the simulation community and their work on ranking and selection.

Computer science The computer science community stumbled into the multiarmed bandit problem in the 1980's, and developed methods that were much simpler than those developed by the applied probability community. This has produced an extensive literature on upper confidence bounding.

7.1.3 The multiarmed bandit story

We would not be doing justice to the learning literature if we did not acknowledge the contribution of a substantial body of literature that addresses what is known as the *multiarmed bandit problem*. The term comes from the common description (in the United States) that a slot machine (in American English), which is sometimes known as a "fruit machine" (in British English), is a "one armed bandit" since each time you pull the arm on the slot machine you are likely to lose money (see figure 7.1).

Now imagine that you have to choose which out of a group of slot machines to play (a surprising fiction since winning probabilities on slot machines are carefully calibrated). Imagine (and this is a stretch) that each slot machine has a different winning probability, and that the only way to learn about the winning probability is to play the machine and observe the winnings. This may mean playing a machine where your estimate of winnings is low, but you acknowledge that your estimate may be wrong, and that you have to try playing the machine to improve your knowledge.

This classic problem has several notable characteristics. The first and most important is the tradeoff between exploration (trying an arm that does not seem to be the best in order to learn more about it) and exploitation (trying arms with higher estimated winnings in order to maximize winnings over time), where winnings are accumulated over time. Other distinguishing characteristics of the basic bandit problem include: discrete choices (that is, slot machines, generally known as "arms"), lookup table belief models (there is a belief about each individual machine), and an underlying process that is stationary (the distribution of winnings does not change over time). Over time, the bandit community has steadily generalized the basic problem.

Multiarmed bandit problems first attracted the attention of the applied probability community in the 1950's, initially in the context of the simpler two-armed problem. It was first formulated as a dynamic program that characterized the optimal policy, but it could not be computed. The multiarmed problem resisted computational solution until the development in 1974 by J.C. Gittins who identified a novel decomposition that led to what are known as *index policies* which involves computing a value ("index") for each arm, and then choosing the arm with the greatest index. While "Gittins indices" (as they came to be known) remain computationally difficult to compute, the elegant simplicity of index policies has guided research into an array of policies that are quite practical.

In 1985, a second breakthrough came from the computer science community, when it was found that a very simple class of policies known as *upper confidence bound* (or UCB) policies (also described below) enjoyed nice theoretical properties in the form of bounds on the number of times that the wrong arm would be visited. The ease with which these policies can be computed (they are a form of index policy) has made them particularly popular in high speed settings such as the internet where there are many situations where it is necessary to make good choices, such as which ad to post to maximize the value of an array of services.

Today, the literature on "bandit problems" has expanded far from its original roots to include any sequential learning problem (which means the state S^n includes a belief state about the function $\mathbb{E}F(x,W)$) where we control the decisions of where to evaluate $F(x,W)$. However, bandit problems now include many problem variations, such as

- Maximizing the final reward rather than just cumulative rewards.
- "Arms" no longer have to be discrete; x may be continuous and vector-valued.
- Instead of one belief about each arm, a belief might be in the form of a linear model that depends on features drawn from x.
- The set of available "arms" to play may change from one round to the next.

The bandit community has fostered a culture of creating problem variations, and then deriving index policies and proving properties (such as regret bounds) that characterize the performance of the policy. While the actual performance of the UCB policies requires careful experimentation and tuning, the culture of creating problem variations is a distinguishing feature of this community. Table 7.1 lists a sampling of these bandit problems, with the original multiarmed bandit problem at the top.

7.1.4 From passive learning to active learning to bandit problems

Chapter 3 describes recursive (or adaptive) learning methods that can be described as a sequence of inputs x^n followed by an observed response y^{n+1}. If we have no control over the inputs x^n, then we would describe this as *passive learning*.

Bandit problem	Description
Multiarmed bandits	Basic problem with discrete alternatives, online (cumulative regret) learning, lookup table belief model with independent beliefs
Best-arm bandits	Identify the optimal arm with the largest confidence given a fixed budget
Restless bandits	Truth evolves exogenously over time
Adversarial bandits	Distributions from which rewards are being sampled can be set arbitrarily by an adversary
Continuum-armed bandits	Arms are continuous
X-armed bandits	Arms are a general topological space
Contextual bandits	Exogenous state is revealed which affects the distribution of rewards
Dueling bandits	The agent gets a relative feedback of the arms as opposed to absolute feedback
Arm-acquiring bandits	New machines arrive over time
Intermittent bandits	Arms are not always available
Response surface bandits	Belief model is a response surface (typically a linear model)
Linear bandits	Belief is a linear model
Dependent bandits	A form of correlated beliefs
Finite horizon bandits	Finite-horizon form of the classical infinite horizon multi-armed bandit problem
Parametric bandits	Beliefs about arms are described by a parametric belief model
Nonparametric bandits	Bandits with nonparametric belief models
Graph-structured bandits	Feedback from neighbors on graph instead of single arm
Extreme bandits	Optimize the maximum of recieved rewards
Quantile-based bandits	The arms are evaluated in terms of a specified quantile
Preference-based bandits	Find the correct ordering of arms

Table 7.1 A sample of the growing population of "bandit" problems.

In this chapter, the inputs x^n are the results of decisions that we make, where it is convenient that the standard notation for the inputs to a statistical model, and decisions for an optimization model, both use x. When we directly control the inputs (that is, we choose x^n), or when decisions influence the inputs, then we would refer to this as *active learning*. Derivative-free stochastic search can always be described as a form of active learning, since we control (directly or indirectly) the inputs which updates a belief model.

At this point you should be asking: what is the difference between derivative-free stochastic search (or as we now know it, active learning) and multiarmed bandit problems? At this stage, we think it is safe to say that the following problem classes are equivalent:

a) Sequential decision problems with a) a dynamic belief state and b) where decisions influence the observations used to update beliefs.

b) Derivative-free stochastic search problems.

c) Active learning problems.

d) Multiarmed bandit problems.

Our position is that problem class (a) is the clearest description of these problems. We note that we are not excluding derivative-based stochastic search in principle. Our presentation of derivative-based stochastic search in chapter 5 did not include any algorithms with a belief state, but we suspect that this will happen in the near future.

A working definition of a bandit problem could be any active learning problem that has been given a label "[adjective]-bandit problem." We claim that *any* sequential decision problem with a dynamic belief state, and where decisions influence the evolution of the belief state, is either a form of bandit problem, or waiting to be labeled as such.

7.2 MODELING DERIVATIVE-FREE STOCHASTIC SEARCH

As with all sequential decision problems, derivative-free stochastic search can be modeled using the five core elements: state variables, decision variables, exogenous information, transition function, and objective function. We first describe each of these five elements in a bit more detail, and then illustrate the model using the context of a problem that involves designing a manufacturing process.

7.2.1 The universal model

Our universal model of any sequential decision problem consists of five elements: state variables, decision variables, exogenous information, the transition function, and the objective function. Below we describe these elements in slightly more detail for the specific context of derivative-free stochastic optimization.

State variables - For derivative-free stochastic optimization, our state variable S^n after n experiments consists purely of the belief state B^n about the function $\mathbb{E}F(x,W)$. In chapter 8 we will introduce problems where we have a physical state R^n such as our budget for making experiments, or the location of a drone collecting information, in which case our state would be $S^n = (R^n, B^n)$. We might have the attributes of a patient in addition to the belief how the patient will respond to a treatment, which gives us a state $S^n = (I^n, B^n)$ (these are often called "contextual problems"), in addition to all three classes of state variables, giving us $S^n = (R^n, I^n, B^n)$. However, this chapter will focus almost exclusively on problems where $S^n = B^n$.

The initial belief B^0 will contain initial estimates of unknown parameters of our belief models. Often, we will have a prior distribution of belief about parameters, in which case B^0 will contain the parameters describing this distribution.

If we do not have any prior information, we will likely have to do some initial exploration, which tends to be guided by some understanding of the problem (especially scaling).

Decision variables - The decision x^n, made after n experiments (which means using the information from S^n), may be binary (do we accept web site A or B), discrete (one of a finite set of choices), continuous (scalar or vector), integer (scalar or vector), and categorical (e.g. the choice of patient type characterized by age, gender, weight, smoker, and medical history).

Decisions are typically made subject to a constraint $x^n \in \mathcal{X}^n$, using a policy that we denote $X^\pi(S^n)$. Here, "π" carries information about the type of function and any tunable parameters. If we run N experiments using policy $X^\pi(S^n)$, we let $x^{\pi,N}$ be the final design. In some cases the policy will be time dependent, in which case we would write it as $X^{\pi,n}(S^n)$.

Most of the time we are going to assume that our decision is to run a single, discrete experiment that returns an observation $W_{x^n}^{n+1}$ or $\hat{F}^{n+1} = F(x^n, W^{n+1})$, but there will be times where x_a^n represents the number of times we run an experiment on "arm" a.

Exogenous information - We let W^{n+1} be the new information that arrives after we choose to run experiment x^n. Often, W^{n+1} is the performance of an experiment, which we would write $W_{x^n}^{n+1}$. More generally, we will write our response function as $F(x^n, W^{n+1})$, in which case W^{n+1} represents observations made that allow us to compute $F(x, W)$ given the decision x. In some cases we may use $F(x^n, W^{n+1})$ to represent the process of running an experiment, where we observe a response $\hat{F}^{n+1} = F(x^n, W^{n+1})$.

Transition function - We denote the transition function by

$$S^{n+1} = S^M(S^n, x^n, W^{n+1}). \tag{7.4}$$

In derivative-free search where S^n is typically the belief about the unknown function $\mathbb{E}F(x, W)$, the transition function represents the recursive updating of statistical model using the methods described in chapter 3. The nature of the updating equations will depend on the nature of the belief model (e.g. lookup table, parametric, neural networks) and whether we are using frequentist or Bayesian belief models.

Objective functions - There are a number of ways to write objective functions in sequential decision problems. Our default notation for derivative-free stochastic search is to let

$F(x, W) =$ the response (could be a contribution or cost, or any performance metric) of running experiment x^n (said differently, running an experiment with design parameters x^n).

Note that $F(x, W)$ is not a function of S_t; we deal with those problems starting in chapter 8.

If we are running a series of experiments in a computer or laboratory setting, we are typically interested in the final design $x^{\pi,N}$, which is a random variable that depends on the initial state S^0 (that may contain a prior distribution of belief B^0) and the experiments $W_{x^0}^1, W_{x^1}^2, \ldots, W_{x^{N-1}}^N$. This means that $x^{\pi,N} = X^\pi(S^N)$ is a random variable. We can evaluate this random variable by running it through a series of tests that we capture with the random variable $\widehat{W}$, which gives us the final-reward objective function

$$\max_\pi \mathbb{E}\{F(x^{\pi,N}, W)|S^0\} = \mathbb{E}_{S^0}\mathbb{E}_{W^1,\ldots,W^N|S^0}\mathbb{E}_{\widehat{W}|S^0}F(x^{\pi,N}, \widehat{W}). \tag{7.5}$$

where $S^0 = B^0$ which is our initial belief about the function.

There are settings where we are running the experiments in the field, and we care about the performance of each of the experiments. In this case, our objective would be the cumulative reward given by

$$\max_{\pi} \mathbb{E}_{S^0}\mathbb{E}_{W^1,\ldots,W^N|S^0}\left\{\sum_{n=0}^{N-1} F(x^n, W^{n+1})|S^0\right\}. \tag{7.6}$$

where $x^n = X^\pi(S^n)$ and where S^0 includes in B^0 anything we know (or believe) about the function before we start.

There are many flavors of performance metrics. We list a few more in section 7.11.1.

We encourage readers to write out all five elements any time you need to represent a sequential decision problem. We refer to this problem as the *base model*. We need this term because later we are going to introduce the idea of a *lookahead model* where approximations are introduced to simplify calculations.

Our challenge, then, is to design effective policies that work well in our base model. We first illustrate this in the context of a classical problem of optimizing a simulation of a manufacturing system.

7.2.2 Illustration: optimizing a manufacturing process

Assume that $x \in \mathcal{X} = \{x_1, \ldots, x_M\}$ represents different configurations for manufacturing a new model of electric vehicle which we are going to evaluate using a simulator. Let $\mu_x = \mathbb{E}_W F(x, W)$ be the expected performance if we could run an infinitely long simulation. We assume that a single simulation (of reasonable duration) produces the performance

$$\hat{F}_x = \mu_x + \varepsilon,$$

where $\varepsilon \sim N(0, \sigma_W^2)$ is the noise from running a single simulation.

Assume we use a Bayesian model (we could do the entire exercise with a frequentist model), where our prior on the truth μ_x is given by $\mu_x \sim N(\bar{\mu}_x^0, \bar{\sigma}_x^{2,0})$. Assume that we have performed n simulations, and that $\mu_x \sim N(\bar{\mu}_x^n, \bar{\sigma}_x^{2,n})$. Our belief B^n about μ_x after n simulations is then given by

$$B^n = (\bar{\mu}_x^n, \bar{\sigma}_x^{2,n})_{x\in\mathcal{X}}. \tag{7.7}$$

For convenience, we are going to define the *precision* of an experiment as $\beta^W = 1/\sigma_W^2$, and the precision of our belief about the performance of configuration x as $\beta_x^n = 1/\bar{\sigma}_x^{2,n}$.

If we choose to try configuration x^n and then run the $n + 1^{st}$ simulation and observe $\hat{F}^{n+1} = F(x^n, W^{n+1})$, we update our beliefs using

$$\bar{\mu}_x^{n+1} = \frac{\beta_x^n \bar{\mu}_x^n + \beta^W \hat{F}_x^{n+1}}{\beta_x^n + \beta^W}, \tag{7.8}$$

$$\beta_x^{n+1} = \beta_x^n + \beta^W, \tag{7.9}$$

if $x = x^n$; otherwise, $\bar{\mu}_x^{n+1} = \bar{\mu}_x^n$ and $\beta_x^{n+1} = \beta_x^n$. These updating equations assume that beliefs are independent; it is a minor extension to allow for correlated beliefs.

We are now ready to state our model using the canonical framework:

State variables The state variable is the belief $S^n = B^n$ given by equation (7.7).

Decision variables The decision variable is the configuration $x \in \mathcal{X}$ that we wish to test next, which will be determined by a policy $X^\pi(S^n)$.

Exogenous information This is the simulated performance given by $\hat{F}^{n+1}(x^n) = F(x^n, W^{n+1})$.

Transition function These are given by equations (7.8)-(7.9) for updating the beliefs.

Objective function We have a budget to run N simulations of different configurations. When the budget is exhausted, we choose the best design according to

$$x^{\pi,N} = \arg\max_{x\in\mathcal{X}} \bar{\mu}_x^N,$$

where we introduce the policy π because $\bar{\mu}_x^N$ has been estimated by running experiments using experimentation policy $X^\pi(S^n)$. The performance of a policy $X^\pi(S^n)$ is given by

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

Our goal is to then solve

$$\max_\pi F^\pi(S^0).$$

This problem called for an objective that optimized the performance of the final design $x^{\pi,N}$, which we call the final reward objective. However, we could change the story to one that involved learning in the field, where we want to optimize as we learn, in which case we would want to optimize the cumulative reward. The choice of objective does not change the analysis approach, but it will change the choice of policy that works best.

7.2.3 Major problem classes

There is a wide range of applications that fall under the umbrella of derivative-free stochastic search. Some of the most important features from the perspective of design policies (which we address next) are:

- Characteristics of the design x - The design variable x may be binary, finite, continuous scalar, vector (discrete or continuous), and multiattribute.
- Noise level - This captures the variability in the outcomes from one experiment to the next. Experiments may exhibit little to no noise, up to tremendously high noise levels, where the noise greatly exceeds the variations among μ_x.
- Time required for an experiment - Experiments can take fractions of a second, seconds, minutes up to hours, weeks, and months.
- Learning budget - Closely related to the time required for an experiment is the budget we have for completing a series of experiments and choosing a design. There are problems where we have a budget of 5,000 observations of ad-clicks to learn the best of 1,000 ads, or a budget of 30 laboratory experiments to learn the best compound out of 30,000.

- Belief model - It helps when we can exploit underlying structural properties when developing belief models. Beliefs may be correlated, continuous (for continuous x), concave (or convex) in x, monotone (outcomes increase or decrease with x). Beliefs may also be Bayesian or frequentist.
- Steady state or transient - It is standard to assume we are observing a process that is not changing over time, but this is not always true.
- Hidden variables - There are many settings where the response depends on variables that we either cannot observe, or simply are not aware of (this may be revealed as a transient process).

The range of problems motivates our need to take a general approach toward designing policies.

7.3 DESIGNING POLICIES

We now turn to the problem of designing policies for either the final reward objective (7.2) or the cumulative reward (7.3). There are two strategies for designing policies, each of which can be further divided into two classes, producing four classes of policies. We provide a brief sketch of these here, and then use the rest of the chapter to give more in-depth examples. It will not be apparent at first, but all four classes of policies will be useful for particular instances of derivative-free stochastic search problems.

Most of the time through this book we use t as our time index, as in x_t and S_t. With derivative-free stochastic search, the most natural indexing is the counter n, as in the n^{th} experiment, observation or iteration. We index the counter n in the superscript (as we first described in chapter 1), which means we have the decision x^n (this is our decision *after* we run our n^{th} experiment), and S^n, which is the information we use to make the decision x^n.

The four classes of polices are given by:

Policy search - Here we use any of the objective functions such as (7.5) or (7.6) to search within a family of functions to find the policy that works best. Policies in the policy-search class can be further divided into two classes:

Policy function approximations (PFAs) - PFAs are analytical functions that map states to actions. They can be lookup tables, or linear models which might be of the form

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

PFAs can also be nonlinear models such as a neural network, although these can require an extremely large number of training iterations.

Cost function approximations (CFAs) - CFAs are parameterized optimization models. A simple one that is widely used in pure learning problems, called interval estimation, is given by

$$X^{CFA-IE}(S^n|\theta^{IE}) = \arg\max_{x\in\mathcal{X}} (\bar{\mu}_x^n + \theta^{IE}\bar{\sigma}_x^n). \tag{7.10}$$

where $\bar{\sigma}_x^n$ is the standard deviation of $\bar{\mu}_x^n$ which declines as the number of times we observe alternative x grows.

A CFA could be a simple sort, as arises with the interval estimation policy in (12.46), but it could also be a linear, nonlinear or integer program, which makes it possible for x to be a large vector instead of one of a discrete set. We can state this generally as

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

where $\bar{C}^\pi(S^n, x|\theta)$ might be a parametrically modified objective function (e.g. with penalties), while $\mathcal{X}^\pi(\theta)$ might be parametrically modified constraints.

Lookahead approximations - An optimal policy can be written as

$$X^{*,n}(S^n) = \arg\max_{x^n}\Bigg(C(S^n, x^n) + \mathbb{E}\left\{\max_\pi \mathbb{E}\left\{\sum_{m=n+1}^{N} C(S^m, X^{\pi,m}(S^m))\middle| S^{n+1}\right\}\middle| S^n, x^n\right\}\Bigg). \quad (7.11)$$

Remember that $S^{n+1} = S^M(S^n, x^n, W^{n+1})$, where there are two potential sources of uncertainty: the exogenous information W^{n+1}, as well as uncertainty about parameters that would be captured in S^n. Remember that for derivative-free stochastic search, the state S^n is our belief state after n observations, which typically consists of continuous parameters (in some cases, vectors of continuous parameters, such as the presence of diseases across countries).

In practice, equation (7.11) cannot be computed, so we have to use approximations. There are two approaches for creating these approximations:

Value function approximations (VFAs) - The ideal VFA policy involves solving Bellman's equation

$$V^n(S^n) = \max_x \big(C(S^n, x) + \mathbb{E}\{V^{n+1}(S^{n+1})|S^n, x\}\big), \quad (7.12)$$

where

$$V^{n+1}(S^{n+1}) = \max_\pi \mathbb{E}\left\{\sum_{m=n+1}^{N} C(S^m, X^{\pi,m}(S^m))\middle| S^{n+1}\right\}.$$

If we could compute this, our optimal policy would be given by

$$X^{*,n}(S^n) = \arg\max_{x\in\mathcal{X}^n} \big(C(S^n, x) + \mathbb{E}\{V^{n+1}(S^{n+1})|S^n, x\}\big). \quad (7.13)$$

Typically we cannot compute $V^{n+1}(S^{n+1})$ exactly. A popular strategy known as "approximate dynamic programming" involves replacing the value function with an approximation $\overline{V}^{n+1}(S^{n+1})$ which gives us

$$X^{VFA,n}(S^n) = \arg\max_{x\in\mathcal{X}^n} \big(C(S^n, x) + \mathbb{E}\{\overline{V}^{n+1}(S^{n+1}|\theta)|S^n, x\}\big). \quad (7.14)$$

Since expectations can be impossible to compute (and approximations are computationally expensive), we often use a value function approximation around the post-decision state, which eliminates the expectation:

$$X^{VFA,n}(S^n) = \arg\max_{x\in\mathcal{X}^n} \big(C(S^n, x) + \overline{V}^{x,n}(S^{x,n}|\theta)\big). \quad (7.15)$$

Direct lookaheads (DLAs) - The second approach is to create an *approximate lookahead model*. If we are making a decision at time t, we represent our lookahead model using the same notation as the base model, but replace the state S^n with $\tilde{S}^{n,m}$, the decision x^n with $\tilde{x}^{n,m}$ which is determined with policy $\tilde{X}^{\tilde{\pi}}(\tilde{S}^{n,m})$, and the exogenous information W^n with $\widetilde{W}^{n,m}$. This creates a *lookahead model* that can be written

$$(S^n, x^n, \widetilde{W}^{n,n+1}, \tilde{S}^{n,n+1}, \tilde{x}^{n,n+1}, \widetilde{W}^{n,n+2}, \ldots, \tilde{S}^{n,m}, \tilde{x}^{n,m}, \widetilde{W}^{n,m+1}, \ldots)$$

We are allowed to introduce any approximations that we think are appropriate for a lookahead model. For example, we may change the belief model, or we may simplify the different types of uncertainty. This gives us an approximate lookahead policy

$$X^{DLA,n}(S^n) = \arg\max_x \Bigg(C(S^n, x) + \tilde{E}\left\{ \max_{\tilde{\pi}} \tilde{E}\left\{ \sum_{m=n+1}^{N} C(\tilde{S}^{n,m}, \tilde{X}^{\tilde{\pi}}(\tilde{S}^{n,m})) | \tilde{S}^{n,n+1} \right\} | S^n, x \right\} \Bigg). \quad (7.16)$$

We emphasize that the lookahead model may be deterministic, but in learning problems the lookahead model has to capture uncertainty. These can be hard to solve, which is why we create a lookahead model that is distinct from the base model which is used to evaluate the policy. We return to lookahead models below.

There are communities in derivative-free stochastic search that focus on *each* of these four classes of policies, so we urge caution before jumping to any conclusions about which class seems best. We emphasize that these are four meta-classes. There are numerous variations within each of the four classes.

We make the claim (backed up by considerable empirical work) that it is important to understand all four classes of policies, given the tremendous variety of problems that we highlighted in section 7.2.3. Finding the best compound out of 3,000 possible choices, with experiments that take 2-4 days to complete, with a budget of 60 days (this is a real problem), is very different than finding the best ads to display to maximize ad-clicks, when we might test 2,000 different ads each day, with millions of views from users. It is inconceivable that we could solve both settings with the same policy.

The next four sections cover each of the four classes of policies:

- Section 7.4 - Policy function approximations
- Section 7.5 - Cost function approximations
- Section 7.6 - Policies based on value function approximations
- Section 7.7 - Policies based on direct lookahead models

After these, sections 7.8, 7.9 and 7.10 provide additional background into two important classes of policies. Section 7.11 discusses evaluating policies, followed by section 7.12 provides some guidance in choosing a policy. We close with a discussion of a series of extensions to our basic model.

7.4 POLICY FUNCTION APPROXIMATIONS

A PFA is any function that maps directly from a state to an action without solving an imbedded optimization problem. PFAs may be any of the function classes we covered in chapter 3, but for pure learning problems, they are more likely to be a parametric function. Some examples include

- An excitation policy - Imagine that demand as a function of price is given by

$$D(p) = \theta_0 - \theta_1 p.$$

We might want to maximize revenue $R(p) = pD(p) = \theta_0 p - \theta_1 p^2$, where we do not know θ_0 and θ_1. Imagine that we have estimates $\bar{\theta}^n = (\bar{\theta}_0^n, \bar{\theta}_1^n)$ after n experiments. Given $\bar{\theta}^n$, the price that optimizes revenue is

$$p^n = \frac{\bar{\theta}_0^n}{2\bar{\theta}_1^n}.$$

After we post price p^n, we observe demand $\hat{D}^{n+1}$, and then use this to update our estimate $\bar{\theta}^n$ using recursive least squares (see section 3.8).

We can learn more effectively if we introduce some noise, which we can do using

$$p^n = \frac{\bar{\theta}_0^n}{2\bar{\theta}_1^n} + \varepsilon^{n+1}. \tag{7.17}$$

where $\varepsilon \sim N(0, \sigma_\varepsilon^2)$, and where the exploration variance σ_ε^2 is a tunable parameter. Let $P^{exc}(S^n|\sigma_\varepsilon)$ represent the excitation policy that determines the price p^n in equation (7.17), parameterized by σ_ε. Also let

$\hat{R}(p^n, \hat{D}^{n+1}) =$ the revenue we earn when we charge price p^n and then observe demand $\hat{D}_{t+1}$.

We tune σ_ε by solving

$$\max_{\sigma_\varepsilon} F(\sigma_\varepsilon) = \mathbb{E} \sum_{n=0}^{N-1} \hat{R}(P^{exc}(S^n|\sigma_\varepsilon), \hat{D}^{n+1}).$$

Excitation policies are quite popular in engineering for learning parametric models. They are ideally suited for online learning, because they favor trying points near the optimum.

- For our pricing problem we derived an optimal price given the belief about demand response, but we could simply pose a linear function of the form

$$X^\pi(S^n|\theta) = \sum_{f \in \mathcal{F}} \theta_f \phi_f(S^n). \tag{7.18}$$

Recall that the recursive formulas provided in section 3.8 imply a state variable given by $S^n = B^n = (\bar{\theta}^n, M^n)$. We now determine θ by solving

$$\max_{\theta} F(\theta) = \mathbb{E} \sum_{n=0}^{N-1} F(X^\pi(S^n|\theta), W^{n+1}). \tag{7.19}$$

We note that $F(\theta)$ is typically highly nonconvex in θ. Algorithms for solving (7.19) remaining an active area of research. We revisit this in chapter 12 when we consider policy function approximations for state-dependent problems.

- Neural networks - While neural networks are growing in popularity as policies, as of this writing we are not aware of their use in pure learning problems as a policy, but this might be an area of research. For example, it is not obvious how to design features $\phi_f(S)$ when the state variable $S^n = (\bar{\theta}^n, M^n)$. A neural network would be able to handle this type of nonlinear response.

  If $X^\pi(S^n|\theta)$ is the neural network and θ is the weight vector (note that θ might have hundreds of thousands to millions of dimensions), the challenge would be to optimize the weights using equation (7.19). Note that the price of this generality is that it would require many iterations to find a good weight vector.

We note in passing that if there is an imbedded optimization problem (which is usually the case) then the policy is technically a form of cost function approximation.

7.5 COST FUNCTION APPROXIMATIONS

Cost function approximations represent what is today one of the most visible and popular classes of learning policies. CFAs describe policies where we have to maximize (or minimize) something to find the alternative to try next, and where we do not make any effort at approximating the impact of a decision now on the future. CFAs cover a wide range of practical, and surprisingly powerful, policies.

Simple greedy policies - We use the term "simple greedy policy" to refer to a policy which chooses an action which maximizes the expected reward given current beliefs, which would be given by

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

Now imagine that we have a nonlinear function $F(x,\theta)$ where θ is an unknown parameter where, after n experiments, might be normally distributed with distribution $N(\theta^n, \sigma^{2,n})$. Our simple greedy policy would solve

$$\begin{aligned} X^{SG}(S^n) &= \arg\max_x F(x,\theta^n), \\ &= \arg\max_x F(x, \mathbb{E}(\theta|S^n)). \end{aligned}$$

This describes a classical approach known under the umbrella as *response surface methods* where we pick the best action based on our latest statistical approximation of a function. We can then add a noise term as we did in our excitation policy in equation (7.17), which introduces a tunable parameter σ_ε.

Bayes greedy - Bayes greedy is just a greedy policy where the expectation is kept on the outside of the function (where it belongs), which would be written

$$X^{BG}(S^n) = \arg\max_x \mathbb{E}_\theta\{F(x,\theta)|S^n\}.$$

When the function $F(x,\theta)$ is nonlinear in θ, this expectation can be tricky to compute. One strategy is to use a sampled belief model and assume that $\theta \in \{\theta_1, \ldots, \theta_K\}$,

and let $p_k^n = Prob[\theta = \theta_k]$ after n iterations. We would then write our policy as

$$X^{BG}(S^n) = \arg\max_x \sum_{k=1}^{K} p_k^n F(x, \theta_k).$$

Finally, we can add a noise term $\varepsilon \sim N(0, \sigma_\varepsilon^2)$, which would then have to be tuned.

Upper confidence bounding - UCB policies, which are very popular in computer science, come in many flavors, but they all share a form that follows one of the earliest UCB policies given by

$$\nu_x^{UCB,n} = \bar{\mu}_x^n + 4\sigma^W \sqrt{\frac{\log n}{N_x^n}}, \tag{7.20}$$

where $\bar{\mu}_x^n$ is our estimate of the value of alternative x, and N_x^n is the number of times we evaluate alternative x within the first n iterations. The coefficient $4\sigma^W$ has a theoretical basis, but is typically replaced with a tunable parameter θ^{UCB} which we might write as

$$\nu_x^{UCB,n}(\theta^{UCB}) = \bar{\mu}_x^n + \theta^{UCB} \sqrt{\frac{\log n}{N_x^n}}. \tag{7.21}$$

The UCB policy, then, would be

$$X^{UCB}(S^n|\theta^{UCB}) = \arg\max_x \nu_x^{UCB,n}(\theta^{UCB}), \tag{7.22}$$

where θ^{UCB} would be tuned using an optimization formulation such as that given in (7.19).

UCB policies all use an index comprised of a current estimate of the value of alternative ("arm" in the language of the bandit-oriented UCB community), given by $\bar{\mu}_x^n$, plus a term that encourages exploration, sometimes called an "uncertainty bonus." As the number of observations grows, $\log n$ also grows (but logarithmically), while N_x^n counts how many times we have sampled alternative x. Note that since initially $N_x^0 = 0$, the UCB policy assumes that we have a budget to try every alternative at least once. When the number of alternatives exceeds the budgets, we either need a prior, or to move away from a lookup table belief model.

Interval estimation - Interval estimation is a class of UCB policy, with the difference that the uncertainty bonus is given by the standard deviation $\bar{\sigma}_x^n$ of the estimate $\bar{\mu}_x^n$ of the value of alternative x. The interval estimation policy is then given by

$$X^{IE}(S^n|\theta^{IE}) = \arg\max_x \left(\bar{\mu}_x^n + \theta^{IE}\bar{\sigma}_x^n\right). \tag{7.23}$$

Here, $\bar{\sigma}_x^n$ is our estimate of the standard deviation of $\bar{\mu}_x^n$. As the number of times we observe action x goes to infinity, $\bar{\sigma}_x^n$ goes to zero. The parameter θ^{IE} is a tunable parameter, which we would tune using equation (7.19).

Thompson sampling - Thompson sampling works by sampling from the current belief about $\mu_x \sim N(\bar{\mu}_x^n, \bar{\sigma}_x^{n,2})$, which can be viewed as the prior distribution for experiment $n+1$. Now choose a sample $\hat{\mu}_x^n$ from the distribution $N(\bar{\mu}_x^n, \bar{\sigma}_x^{n,2})$. The Thompson sampling policy is then given by

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

Thompson sampling is more likely to choose the alternative x with the largest $\bar{\mu}_x^n$, but because we sample from the distribution, we may also choose other alternatives, but are unlikely to choose alternatives where the estimate $\bar{\mu}_x^n$ is low relative to the others.

Note that we can create a tunable version of Thompson sampling by choosing $\hat{\mu}_x^n \sim N(\bar{\mu}_x^n, (\theta^{TS}\bar{\sigma}_x^n)^2)$, in which case we would write our policy as $X^{TS}(S^n|\theta^{TS})$. Now we just have to tune θ^{TS}.

Boltzmann exploration - A different form of maximizing over actions involves computing a probability that we pick an action x, given an estimate $\bar{\mu}_x^n$ of the reward from this action. This is typically computed using

$$p^n(x|\theta) = \frac{e^{\theta\bar{\mu}_x^n}}{\sum_{x'} e^{\theta\bar{\mu}_{x'}^n}}. \tag{7.24}$$

Now pick x^n at random according to the distribution $p^n(x|\theta)$. Boltzmann exploration is sometimes referred to as "soft max" since it is performing a maximization in a probabilistic sense.

Both PFAs and CFAs require tuning a parameter θ (which is often a scalar but may be a vector), where the tuning can be used to maximize either a final reward or cumulative reward objective function. We note that searching for θ is its own sequential decision problem, which requires a policy that we would call a *learning policy* (finding θ) that then produces a good *implementation policy* $X^\pi(S^n|\theta)$.

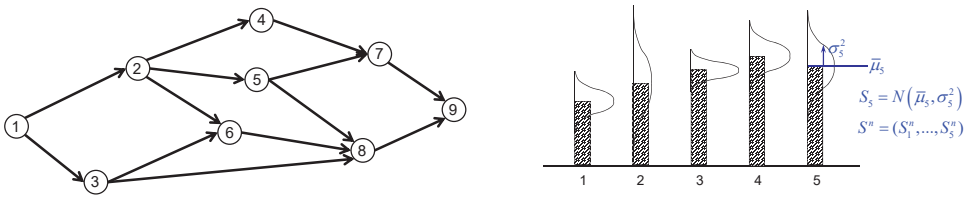
7.6 VFA-BASED POLICIES

A powerful algorithmic strategy for some problem classes is based on Bellman's equation, which we introduced briefly in section 2.1.3. This approach has received less attention in the context of learning problems, with the notable exception of one community that is centered on the idea known as "Gittins indices" which we review below. Gittins indices, popular in the applied probability community, are virtually unheard of in computer science which focuses on upper confidence bounding. Gittins indices are much harder to compute (as we show below), but it was Gittins indices that introduced the idea of using an index policy that was the original inspiration for UCB policies (this connection occurred in 1984, 10 years after the first paper on Gittins indices).

In this section, we are going to begin in section 7.6.1 by introducing the general idea of using Bellman's equation for pure learning problems. Section 7.6.2 will illustrate the ideas in the context of a simple problem where we are testing a new drug, where observations are all 0 or 1. Then, section 7.6.3 will introduce a powerful approximation strategy based on the idea of approximating value functions. We close in section 7.6.4 by covering the rich history and theory behind Gittins indices, which laid the foundation for modern research in pure learning problems.

7.6.1 An optimal policy

Consider the graph in figure 7.2(a). Imagine that our state (that is, the node where we are located) is $S^n = 2$, and we are considering a decision $x_s = 5$ that puts us in state $S^{n+1} = 5$. Let $\mathcal{X}^n$ be the states (nodes) we can reach from state (node) S^n, and assume



(a) (b)

Figure 7.2 (a) Optimizing over a graph, where we are considering the transition from node (state) 2 to node (state) 5. (b) Optimizing a learning problem, where we are considering evaluating alternative 5, which would change our belief (state) $S_5^n = (\bar{\mu}_x^n, \sigma_5^{2,n})$ to belief (state) $S_5^{n+1} = (\bar{\mu}_x^{n+1}, \sigma_5^{2,n+1})$.

that we have a value $V^{n+1}(s')$ for each $s \in \mathcal{X}^n$. Then we can write the value of being in state S^n using Bellman's equation, which gives us

$$V^n(S^n) = \max_{s' \in \mathcal{X}^n} \big(C(S^n, s') + V^{n+1}(s')\big). \tag{7.25}$$

Bellman's equation, as given in equation (7.25), is fairly intuitive. In fact, this is the foundation of every shortest path algorithm that is used in modern navigation systems.

Now consider the learning problem in figure 7.2(b). Our state is our belief about the true performance of each of the five alternatives. Recall that the precision $\beta_x^n = 1/\sigma_x^{2,n}$. We can express our state as $S^n = (\bar{\mu}_x^n, \beta_x^n)_{x \in \mathcal{X}}$. Now imagine we decide to experiment with the 5^{th} alternative, which will give us an observation W_5^{n+1}. The effect of our observation W_5^{n+1} will take us to state

$$\bar{\mu}_5^{n+1} = \frac{\beta_5^n \bar{\mu}_5^n + \beta^W W_5^{n+1}}{\beta_5^n + \beta^W}, \tag{7.26}$$

$$\beta_5^{n+1} = \beta_5^n + \beta^W. \tag{7.27}$$

The values for $\bar{\mu}_x^n$ and β_x^n for x other than 5 are unchanged.

The only differences between our graph problem and our learning problem are:

a) The decision to move from state (node) 2 to state 5 in the graph problem is a deterministic transition.

b) The states in our graph problem are discrete, while the state variables in the learning problem are continuous and vector valued.

In our learning problem, we make a decision $x^n = 5$ to test alternative 5, but the outcome W_5^{n+1} is random, so we do not know what state the experiment will take us to. However, we can fix this by inserting an expectation in Bellman's equation, giving us

$$V^n(S^n) = \max_{x \in \mathcal{X}} \big(C(S^n, x) + \mathbb{E}_{S^n} \mathbb{E}_{W|S^n} \{V^{n+1}(S^{n+1}) | S^n, x\}\big), \tag{7.28}$$

where the first expectation $\mathbb{E}_{S^n}$ handles the uncertainty in the true value μ_x given the belief in S^n, while the second expectation $\mathbb{E}_{W|S^n}$ handles the noise in the observation $W_x^{n+1} = \mu_x + \varepsilon^{n+1}$ of our unknown truth μ_x. Note that if we are using a frequentist belief model, we would just use $\mathbb{E}_W$.

Other than the expectation in equation (7.28), equations (7.25) and (7.28) are basically the same. The point is that we can use Bellman's equation regardless of whether the state is a node in a network, or the belief about the performance of a set of alternatives. State variables are state variables, regardless of their interpretation.

If we could solve equation (7.28), we would have an optimal policy given by

$$X^*(S^n) = \arg\max_{x\in\mathcal{X}} \big(C(S^n,x) + \mathbb{E}_{S^n}\mathbb{E}_{W|S^n}\{V^{n+1}(S^{n+1})|S^n,x\}\big). \tag{7.29}$$

Equation (7.28) is set up for problems which maximize undiscounted cumulative rewards. If we want to solve a final reward problem, we just have to ignore contributions until the final evaluation, which means we write Bellman's equation as

$$V^n(S^n) = \max_{x\in\mathcal{X}} \begin{cases} \big(0 + \mathbb{E}_{S^n}\mathbb{E}_{W|S^n}\{V^{n+1}(S^{n+1})|S^n,x\}\big) & n < N, \\ C(S^N,x) & n = N. \end{cases} \tag{7.30}$$

We could also change our objective to a discounted, infinite horizon model by simply adding a discount factor $\gamma < 1$, which changes equation (7.28) to

$$V^n(S^n) = \max_{x\in\mathcal{X}} \big(C(S^n,x) + \gamma\mathbb{E}_W\{V^{n+1}(S^{n+1})|S^n,x\}\big). \tag{7.31}$$

The problem with Bellman's equation is that while it is not hard finding the value of being at each node in a graph (even if there are 100,000 nodes), handling a belief state is much harder. If we have a problem with just 20 alternatives, the state $S^n = (\bar{\mu}^n_x, \beta^n_x)_{x\in\mathcal{X}}$ would have 40 continuous dimensions, which is an extremely difficult estimation problem given noisy measurements and reasonable computing budgets. There are real applications with thousands of alternatives (or more), as would occur when finding the best out of thousands of compounds to fight a disease, or the best news article to display on the website of a news organization.

7.6.2 Beta-Bernoulli belief model

There is an important class of learning problem that can be solved exactly. Imagine that we are trying to learn whether a new drug is successful. We are testing patients, where each test yields the outcome of success ($W^n = 1$) or failure ($W^n = 0$). Our only decision is the set

$$\mathcal{X} = \{continue, patent, cancel\}.$$

The decision to "patent" means to stop the trial and file for a patent on the drug as the step before going to market. The decision to "cancel" means to stop the trial and cancel the drug. We maintain a state variable R^n where

$$R^n = \begin{cases} 1 & \text{if we are still testing,} \\ 0 & \text{if we have stopped testing.} \end{cases}$$

The evolution of R^n is given by

$$R^{n+1} = \begin{cases} 1 & R^n = 1 \text{ and } x^n = \text{"continue"}, \\ 0 & \text{otherwise.} \end{cases}$$

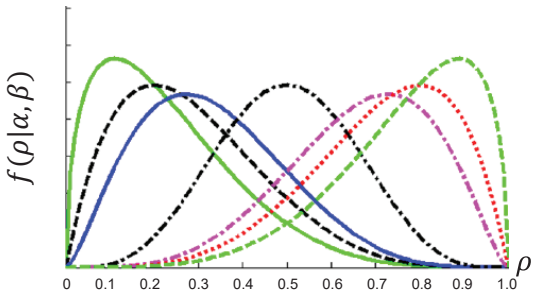


Figure 7.3 A family of densities from a beta distribution.

As the experiments progress, we are going to keep track of successes α^n and failures β^n using

$$\begin{aligned}\alpha^{n+1} &= \alpha^n + W^{n+1},\\ \beta^{n+1} &= \beta^n + (1 - W^{n+1}).\end{aligned}$$

We can then estimate the probability of success of the drug using

$$\rho^n = \frac{\alpha^n}{\alpha^n + \beta^n}.$$

The state variable is

$$S^n = (R^n, \alpha^n, \beta^n).$$

We can create a belief about the true probability of success, ρ, by assuming that it is given by a beta distribution with parameters (α^n, β^n) which is given by

$$f(\rho|\alpha,\beta) = \frac{\Gamma(\alpha)}{\Gamma(\alpha) + \Gamma(\beta)}\rho^{\alpha-1}(1-\rho)^{\beta-1}$$

where $\Gamma(k) = k!$. The beta distribution is illustrated in figure 7.3. Given we are in state S^n, we then assume that $\rho^{n+1} \sim beta(\alpha^n, \beta^n)$.

This is a model that allows us to compute Bellman's equation in (7.28) exactly (for finite horizon problems) since R^n, α^n, and β^n are all discrete. We have to specify the negative cost of continuing (that is, the cost of running the study), along with the contribution of stopping to patent, versus canceling the drug.

7.6.3 Backward approximate dynamic programming

Relatively little attention has been directed to learning problems from the communities that use the concepts of approximate dynamic programming. Readers will not see these ideas in this book until chapters 15, 16, and 17. To avoid duplication we are just going to sketch how we might use what is becoming known as backward approximate dynamic programming, which we introduce in greater depth in chapter 15.

Assume for the moment that we have a standard discrete learning model where we are trying to learn the true values μ_x for each $x \in \mathcal{X} = \{x_1, \ldots, x_M\}$. Assume our beliefs are normally distributed, which means that after n experiments,

$$\mu_x \sim N(\bar{\mu}_x^n, \bar{\sigma}_x^{2,n}).$$

Our belief state, then, would be

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

What we are going to do is to replace our value function $V^n(S^n)$ with a linear model of the form

$$V^n(S^n|\theta^n) \approx \overline{V}^n(S^n) = \sum_{f \in \mathcal{F}} \theta_f^n \phi_f(S^n). \tag{7.32}$$

Note that we are making a point of modeling our problem as a finite horizon problem, which means that our value function approximation $\overline{V}^n(S^n)$ depends on the index n, which is why we had to index the vector $\theta^n = (\theta_f^n)_{f \in \mathcal{F}}$.

The features $(\phi_f)_{f \in \mathcal{F}}$ (which do not depend on n) are features that we have designed from our state variable S^n. For example, imagine for our features that we sort the alternatives based on the basis of the index ν_x^n given by

$$\nu_x^n = \bar{\mu}_x^n + 2\bar{\sigma}_x^n.$$

Now assume the alternatives are sorted so that $\nu_{x_1}^n \geq \nu_{x_2}^n \geq \ldots \geq \nu_{x_F}^n$, where F might be the top 20 alternatives (this sorting is very important - the sorting has to be done at every iteration). Next create features such as

$$\begin{aligned}
\phi_{x,1} &= \bar{\mu}_x^n, \\
\phi_{x,2} &= (\bar{\mu}_x^n)^2, \\
\phi_{x,3} &= \bar{\sigma}_x^n, \\
\phi_{x,4} &= \bar{\mu}_x^n \bar{\sigma}_x^n, \\
\phi_{x,5} &= \bar{\mu}_x^n + 2\bar{\sigma}_x^n.
\end{aligned}$$

Now we have five features per alternative, times 20 alternatives which gives us a model with 100 features (and 101 parameters, when we include a constant term).

Backward approximate dynamic programming works roughly as follows:

Step 0 Set $\theta^{N+1} = 0$, giving us $\overline{V}^{N+1}(S^{N+1}) = 0$. Set $n = N$.

Step 1 Sample K states from the set of possible values of the state S. For each sampled state $\hat{s}_k^n$, compute an estimated value $\hat{v}_k^n$ from

$$\hat{v}_k^n = \max_x \big(C(\hat{s}_k^n, x) + \mathbb{E}\{\overline{V}^{n+1}(S^{n+1}|\theta^{n+1})|\hat{s}_k^n, x\}\big), \tag{7.33}$$

where S^{n+1} is computed by sampling what we might observe W_x^{n+1} from choosing to test alternative x, and where $\overline{V}^{n+1}(S^{n+1}|\theta^{n+1})$ is given by our approximation in equation (7.32). The expectation has to average over these outcomes.

Step 2 Take the set of values $(\hat{s}_k^n, \hat{v}_k^n)_{k=1}^K$ and fit a new linear model $\overline{V}^n(s|\theta^n)$ using batch linear regression (see section 3.7.1).

Step 3 Set $n \leftarrow n-1$. If $n \geq 0$, return to Step 1.

This approach is relatively immune to the dimensionality of the state variable or exogenous information variable. In fact, we can even use these ideas if the belief is expressed using a parametric model, although we would have to design a new set of features. Finally, the logic does not even require that the alternatives x be discrete, since we can think of solving equation (7.33) as a nonlinear programming problem. However, this idea has seen only minimal experimentation.

We suspect that a VFA-based policy is best suited for problems with small learning budgets. Note that the resulting policy is nonstationary (it depends on n), in contrast with the CFA-based policies that are so popular in some communities for their simplicity.

We emphasize the idea of using value function approximations for learning models is quite young, and at this stage we offer no guarantees on the performance of this particular approximation strategy. We present this logic to illustrate the idea that we may be able to solve the so-called "curse of dimensionality" of dynamic programming using statistical models.

7.6.4 Gittins indices for learning in steady state

We are going to finally turn to what was viewed as a breakthrough result in the 1970's. "Bandit problems" were initially proposed in the 1950's, with some theoretical interest in the 1970's. The idea of characterizing an optimal policy using Bellman's equation (as in equation (7.28)) first emerged during this time. The idea of using the belief as a state variable was a central insight. However, solving Bellman's equation looked completely intractable.

Basic idea - In 1974, John Gittins introduced the idea of decomposing bandit problems using what is known as "Lagrangian relaxation." In a nutshell, the bandit problem requires that we observe one, and only one, alternative at a time. If we write our decision as $x_i^n = 1$ if we choose to test alternative i, then we would introduce a constraint

$$\sum_{i=1}^{M} x_i^n = 1. \tag{7.34}$$

Now imagine solving our optimization problem where we relax the constraint (7.34). Once we do this, the problem decomposes into M dynamic programs (one for each arm). However, we put a price, call it ν^n, on the constraint (7.34), which means we would write our optimization problem as

$$\min_{(\nu^n)_{n=1}^N} \max_{(\pi^n)_{n=1}^N} \mathbb{E}\left\{\sum_{n=0}^{N}\left(W_{x^n}^{n+1} + \nu^n\left(\sum_{i=1}^{M} x_i^n - 1\right)\right)|S^0\right\}, \tag{7.35}$$

where $x^n = X^{\pi^n}(S^n)$ is the policy at iteration n.

This problem looks complicated, because we have to find a policy $X^{\pi^n}(S^n)$ for each iteration n, and we also have to optimize the penalty (also known as a dual variable or shadow price) ν^n for each n. But what if we solve the steady state version of the problem, given by

$$\min_{\nu} \max_{\pi} \mathbb{E}\left\{\sum_{n=0}^{\infty}\gamma^n\left(W_{x^n}^{n+1} + \nu\left(\sum_{i=1}^{M} x_i^n - 1\right)\right)|S^0\right\}. \tag{7.36}$$

where $x^n = X^\pi(S^n)$ is our now stationary policy, and there is a single penalty ν. For the infinite horizon problem we have to introduce a discount factor γ (we could do this for the finite horizon version, but this is generally not necessary).

Now the problem decomposes by alternative. For these problems, we now choose between continuing to test, or stopping. When we continue testing alternative x, we not only receive W_x^{n+1}, we also pay a "penalty" ν. A more natural way to formulate the problem is to assume that if you continue to play, you receive the reward W_x^{n+1}, whereas if you stop you receive a reward $r = -\nu$.

We now have the following dynamic program for each arm where the decision is only whether to "Stop" or "Continue," which gives us

$$V_x(S|r) = \max\{\underbrace{r + \gamma V_x(S|r)}_{Stop}, \underbrace{\mathbb{E}_W\{W_x + \gamma V_x(S'|r)|S^n\}}_{Continue}\}, \tag{7.37}$$

where S' is the updated state given the random observation W whose distribution is given by the belief in S^n. For example, if we have binomial (0/1) outcomes, S^n would be the probability ρ^n that $W = 1$ (as we did in section 7.6.2). For normally distributed rewards, we would have $\mathbb{E}_{W|S^n} = \bar{\mu}^n$. If we stop, we do not learn anything so the state S stays the same.

It can be shown that if we choose to stop sampling in iteration n and accept the fixed payment ρ, then that is the optimal strategy for all future rounds. This means that starting at iteration n, our optimal future payoff (once we have decided to accept the fixed payment) is

$$\begin{aligned} V(S|r) &= r + \gamma r + \gamma^2 r + \cdots \\ &= \frac{r}{1-\gamma}, \end{aligned}$$

which means that we can write our optimality recursion in the form

$$V(S^n|r) = \max\left[\frac{r}{1-\gamma}, \bar{\mu}^n + \gamma\mathbb{E}\left\{V(S^{n+1}|r)\middle| S^n\right\}\right]. \tag{7.38}$$

Now for the magic of Gittins indices. Let Γ be the value of r which makes the two terms in the brackets in (7.38) equal (the choice of Γ is in honor of Gittins). That is,

$$\frac{\Gamma}{1-\gamma} = \mu + \gamma\mathbb{E}\left\{V(S|\Gamma)|\, S\right\}. \tag{7.39}$$

The hard part of Gittins indices is that we have to iteratively solve Bellman's equation for different values of Γ until we find one where equation (7.39) is true. The reader should conclude from this that Gittins indices are computable (this is the breakthrough), but computing them is not easy.

We assume that W is random with a known variance σ_W^2. Let $\Gamma^{Gitt}(\mu, \sigma, \sigma_W, \gamma)$ be the solution of (7.39). The optimal solution depends on the current estimate of the mean, μ, its variance σ^2, the variance of our measurements σ_W^2, and the discount factor γ. For notational simplicity, we are assuming that the experimental noise σ_W^2 is independent of the action x, but this assumption is easily relaxed.

Next assume that we have a set of alternatives $\mathcal{X}$, and let $\Gamma_x^{Gitt,n}(\bar{\mu}_x^n, \bar{\sigma}_x^n, \sigma_W, \gamma)$ be the value of Γ that we compute for each alternative $x \in \mathcal{X}$ given state $S^n = (\bar{\mu}_x^n, \bar{\sigma}_x^n)_{x\in\mathcal{X}}$. An

optimal policy for selecting the alternative x is to choose the one with the highest value for $\Gamma_x^{Gitt,n}(\bar{\mu}_x^n, \bar{\sigma}_x^n, \sigma_W, \gamma)$. That is, we would make our choice using

$$\max_x \Gamma_x^{Gitt,n}(\bar{\mu}_x^n, \bar{\sigma}_x^n, \sigma_W, \gamma).$$

Such policies are known as *index policies*, which refers to the property that the parameter $\Gamma_x^{Gitt,n}(\bar{\mu}_x^n, \bar{\sigma}_x^n, \sigma_W, \gamma)$ for alternative x depends only on the characteristics of alternative x. For this problem, the parameters $\Gamma_x^{Gitt,n}(\bar{\mu}_x^n, \bar{\sigma}_x^n, \sigma_W, \gamma)$ are called Gittins indices. While Gittins indices have attracted little attention outside the probability community (given the computational complexity), the concept of index policies captured the attention of the research community in 1984 in the first paper that introduced upper confidence bounding (in our CFA class). So, the real contribution of Gittins index policies is the simple idea of an index policy.

We next provide some specialized results when our belief is normally distributed.

Gittins indices for normally distributed rewards - Students learn in their first statistics course that normally distributed random variables enjoy a nice property. If Z is normally distributed with mean 0 and variance 1 and if

$$X = \mu + \sigma Z$$

then X is normally distributed with mean μ and variance σ^2. This property simplifies what are otherwise difficult calculations about probabilities of events.

The same property applies to Gittins indices. Although the proof requires some development, it is possible to show that

$$\Gamma^{Gitt,n}(\bar{\mu}^n, \bar{\sigma}^n, \sigma_W, \gamma) = \mu + \Gamma(\frac{\bar{\sigma}^n}{\sigma_W}, \gamma)\sigma_W,$$

where

$$\Gamma(\frac{\bar{\sigma}^n}{\sigma_W}, \gamma) = \Gamma^{Gitt,n}(0, \sigma, 1, \gamma)$$

is a "standard normal Gittins index" for problems with mean 0 and variance 1. Note that $\bar{\sigma}^n/\sigma_W$ decreases with n, and that $\Gamma(\frac{\bar{\sigma}^n}{\sigma_W}, \gamma)$ decreases toward zero as $\bar{\sigma}^n/\sigma_W$ decreases. As $n \to \infty$, $\Gamma^{Gitt,n}(\bar{\mu}^n, \bar{\sigma}^n, \sigma_W, \gamma) \to \bar{\mu}^n$.

Unfortunately, as of this writing, there do not exist easy-to-use software utilities for computing standard Gittins indices. Table 7.2 is exactly such a table for Gittins indices. The table gives indices for both the variance-known and variance-unknown cases, but only for the case where $\frac{\sigma^n}{\sigma_W} = \frac{1}{n}$. In the variance-known case, we assume that σ^2 is given, which allows us to calculate the variance of the estimate for a particular slot machine just by dividing by the number of observations.

Lacking standard software libraries for computing Gittins indices, researchers have developed simple approximations. As of this writing, the most recent of these works as follows. First, it is possible to show that

$$\Gamma(s, \gamma) = \sqrt{-\log \gamma} \cdot b\left(-\frac{s^2}{\log \gamma}\right). \tag{7.40}$$

A good approximation of $b(s)$, which we denote by $\tilde{b}(s)$, is given by

$$\tilde{b}(s) = \begin{cases} \frac{s}{\sqrt{2}} & s \leq \frac{1}{7}, \\ e^{-0.02645(\log s)^2 + 0.89106 \log s - 0.4873} & \frac{1}{7} < s \leq 100, \\ \sqrt{s}\left(2\log s - \log\log s - \log 16\pi\right)^{\frac{1}{2}} & s > 100. \end{cases}$$

	Discount factor			
	Known variance		Unknown variance	
Observations	0.95	0.99	0.95	0.99
1	0.9956	1.5758	-	-
2	0.6343	1.0415	10.1410	39.3343
3	0.4781	0.8061	1.1656	3.1020
4	0.3878	0.6677	0.6193	1.3428
5	0.3281	0.5747	0.4478	0.9052
6	0.2853	0.5072	0.3590	0.7054
7	0.2528	0.4554	0.3035	0.5901
8	0.2274	0.4144	0.2645	0.5123
9	0.2069	0.3808	0.2353	0.4556
10	0.1899	0.3528	0.2123	0.4119
20	0.1058	0.2094	0.1109	0.2230
30	0.0739	0.1520	0.0761	0.1579
40	0.0570	0.1202	0.0582	0.1235
50	0.0464	0.0998	0.0472	0.1019
60	0.0392	0.0855	0.0397	0.0870
70	0.0339	0.0749	0.0343	0.0760
80	0.0299	0.0667	0.0302	0.0675
90	0.0267	0.0602	0.0269	0.0608
100	0.0242	0.0549	0.0244	0.0554

Table 7.2 Gittins indices $\Gamma(\frac{\sigma^n}{\sigma_W}, \gamma)$ for the case of observations that are normally distributed with mean 0, variance 1, and where $\frac{\sigma^n}{\sigma_W} = \frac{1}{n}$ (from Gittins (1989)).

Thus, the approximate version of (7.40) is

$$\Gamma^{Gitt,n}(\mu, \sigma, \sigma_W, \gamma) \approx \bar{\mu}^n + \sigma_W \sqrt{-\log \gamma} \cdot \tilde{b} \left(-\frac{\bar{\sigma}^{2,n}}{\sigma_W^2 \log \gamma} \right). \tag{7.41}$$

Comments - While Gittins indices were considered a major breakthrough, it has largely remained an area of theoretical interest in the applied probability community. Some issues that need to be kept in mind when using Gittins indices are:

- While Gittins indices were viewed as a computational breakthrough, they are not, themselves, easy to compute.

- Gittins index theory only works for infinite horizon, discounted, cumulative reward problems. Gittins indices are not optimal for finite horizon problems which is what we always encounter in practice, but Gittins indices may still be a useful approximation.

- Gittins theory is limited to lookup table belief models (that is, discrete arms/alternatives) with independent beliefs. This is a major restriction in real applications.

We note that Gittins indices are not widely used in practice, but it was the development of Gittins indices that established the idea of using index policies, which laid the foundation for all the work on upper confidence bounding, first developed in 1984, but which has seen explosive growth post 2000, largely driven by search algorithms on the internet.

7.7 DIRECT LOOKAHEAD POLICIES

There are certain classes of learning problems that require that we actually plan into the future, just as a navigation package will plan a path to the destination. The difference is that while navigation systems can get away with solving a deterministic approximation, recognizing and modeling uncertainty is central to learning problems.

We are going to begin in section 7.7.1 with a discussion of the types of learning problems where a lookahead policy is likely to add value. Then, we are going to describe a series of lookahead strategies that progress in stages before going to a full stochastic lookahead. These will be presented as follows:

- Section 7.7.2 discusses the powerful idea of using one-step lookaheads, which is very useful for certain problem classes where experiments are expensive.

- When one-step lookaheads do not work, a useful strategy is to do a restricted, multistep lookahead, which we describe in section 7.7.3.

- Section 7.7.4 proposes a full multiperiod, deterministic lookahead.

- Section 7.7.5 illustrates a full, multiperiod stochastic lookaheads, although a proper discussion is given in chapter 19.

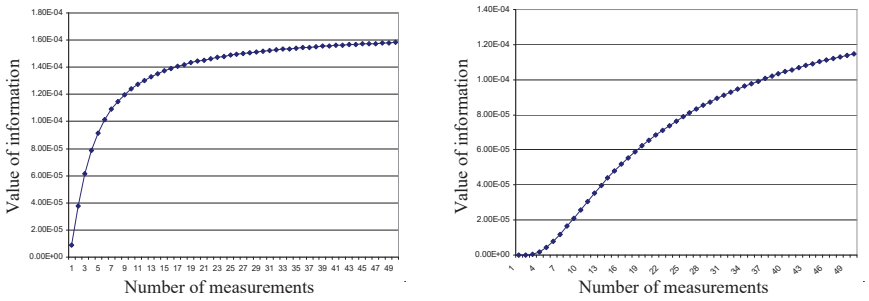
- Section 7.7.6 describes a class of hybrid policies.

7.7.1 When do we need lookahead policies?

Lookahead policies are very important when we are managing a physical resource. That is why navigation systems have to plan a path to the destination in order to figure out what to do now. By contrast, the most popular policies for pure learning problems are in the CFA class such as upper confidence bounding, Thompson sampling and interval estimation.

However, there are pure learning problems where different classes of lookahead policies are particularly useful. As we progress through our direct lookahead policies, the following problem characteristics will prove to be important:

- Complex belief models - Imagine testing the market response to charging a price $p = \$100$ for a book, and we find that sales are higher than expected. Then we would expect that prices $p = \$95$ and $p = \$105$ would also be higher than expected. This reflects correlated beliefs. Direct lookahead models can capture these interactions,



7.4(a) 7.4(b)

Figure 7.4 Value of making n observations. In (a), the value of information is concave, while in (b) the value of information follows an S-curve.

while pure index policies such as upper confidence bounding and Thompson sampling cannot (although they pick up these effects indirectly through the tuning process).

- Expensive experiments/small budgets - There are many settings where experiments are expensive, which means that we will have budgets (typically for both time and money) that limit how many we can do. With a limited budget, we will be more interested in exploring in the early experiments than toward the end. This is particularly pronounced with cumulative rewards, but it is also true with final rewards.

- Noisy experiments/S-curve value of information - There are problems that enjoy the intuitive behavior where the value of repeating the same experiment multiple times provides increasing value, but with decreasing marginal returns, as illustrated in figure 7.4(a). When the noise from running an experiment is high enough, the marginal value of running an experiment can actually grow, as illustrated in figure 7.4(b).

  The S-curve behavior in figure 7.4(b) arises when experiments are noisy, which means that a single experiment contributes little information. This behavior is actually quite common, especially when the outcome of an experiment is a success or failure (perhaps indicated by 1 or 0).

  When we have an S-curve value of information, this means we have to think about how many times we can evaluate an alternative x. It may take 10 repetitions before we are learning anything. If we have 100 alternatives and a budget of 50, we are simply not going to be able to evaluate each alternative 10 times, which means we are going to have to completely ignore a number of alternatives. However, making this decision requires planning into the future given the experimental budget.

- Large number of alternatives relative to the budget - There are problems where the number of alternatives to test is much larger than our budget. This means that we are simply not going to be able to do a good job evaluating alternatives. We would need to plan into the future to know if it is worth trying even one experiment. This is most pronounced when the value of information follows an S-curve, but arises even when the value of information is concave.

7.7.2 Single period lookahead policies

A single period lookahead would never work well if we had to deal with a physical state (imagine solving a shortest path problem over a graph with a single period lookahead). However, they often work exceptionally well in the setting of learning problems. Some approaches for performing single-period lookaheads are given below:

Knowledge gradient for final reward - The most common form of single-period lookahead policy are value-of-information policies, which maximize the value of information from a single experiment. Let $S^n = (\bar{\mu}_x^n, \beta_x^n)_{x\in\mathcal{X}}$ be our belief state now where $\bar{\mu}_x^n$ is our estimate of the performance of design x, and β_x^n is the precision (one over the variance).

Imagine that we are trying to find the design x that maximizes μ_x, where μ_x is unknown. Let $\bar{\mu}_x^n$ be our best estimate of μ given our state of knowledge (captured by $S^n = B^n$). If we stopped now, we would choose our design x^n by solving

$$x^n = \arg\max_{x'} \bar{\mu}_{x'}^n.$$

Now imagine running experiment $x = x^n$, where we will make a noisy observation

$$W_{x^n}^{n+1} = \mu_{x^n} + \varepsilon^{n+1},$$

where $\varepsilon^{n+1} \sim N(0, \sigma_W^2)$. This will produce an updated estimate $\bar{\mu}_{x'}^{n+1}(x^n)$ for $x' = x^n$ which we compute using

$$\bar{\mu}_{x^n}^{n+1} = \frac{\beta_{x^n}^n \bar{\mu}_{x^n}^n + \beta^W W_{x^n}^{n+1}}{\beta_{x^n}^n + \beta^W}, \tag{7.42}$$

$$\beta_{x^n}^{n+1} = \beta_{x^n}^n + \beta^W. \tag{7.43}$$

For $x' \neq x^n$, $\bar{\mu}_{x'}^{n+1}$ and $\beta_{x'}^{n+1}$ are unchanged. This gives us an updated state $S^{n+1}(x) = (\bar{\mu}_{x'}^{n+1}, \beta_{x'}^{n+1})_{x'\in\mathcal{X}}$ which is random because we have not yet observed W_x^{n+1} (we are still trying to decide if we should run experiment x).

The value of our solution after this experiment (given what we know at time n) is given by

$$\mathbb{E}_{S^n}\mathbb{E}_{W|S^n}\{\max_{x'} \bar{\mu}^{n+1}(x))|S^n\}.$$

We can expect that our experiment using parameters (or design) x would improve our solution, so we can evaluate this improvement using

$$\nu^{KG}(x) = \mathbb{E}_{S^n}\mathbb{E}_{W|S^n}\{\max_{x'} \bar{\mu}^{n+1}(x))|S^n\} - \max_{x'} \bar{\mu}_{x'}^n. \tag{7.44}$$

The quantity $\nu^{KG}(x)$ is known as the *knowledge gradient*, and it gives the expected value of the information from experiment x. This calculation is made by looking one experiment into the future. We cover knowledge gradient policies in considerably greater depth in section 7.8.

Expected improvement - Known as EI in the literature, expected improvement is a close relative of the knowledge gradient, given by the formula

$$\nu_x^{EI,n} = \mathbb{E}\left[\max\left\{0, \mu_x - \max_{x'} \bar{\mu}_{x'}^n\right\} \middle| S^n, x = x^n\right]. \tag{7.45}$$

Unlike the knowledge gradient, EI does not explicitly capture the value of an experiment, which requires evaluating the ability of an experiment to change the final design decision. Rather, it measures the degree to which an alternative x *might* be better. It does this by capturing the degree to which the random truth μ_x *might* be greater than the current best estimate $\max_{x'} \bar{\mu}^n_{x'}$.

Sequential kriging - This is a methodology developed in the geosciences to guide the investigation of geological conditions, which are inherently continuous and two- or three-dimensional. Kriging evolved in the setting of geo-spatial problems where x is continuous (representing a spatial location, or even a location underground in three dimensions). For this reason, we let the truth be the function $\mu(x)$, rather than μ_x (the notation we used when x was discrete).

Kriging uses a form of meta-modeling where the surface is assumed to be represented by a linear model, a bias model and a noise term which can be written as

$$\mu(x) = \sum_{f\in\mathcal{F}} \theta_f \phi_f(x) + Z(x) + \varepsilon,$$

where $Z(x)$ is the bias function and $(\phi_f(x))_{f\in\mathcal{F}}$ are a set of features extracted from data associated with x. Given the (assumed) continuity of the surface, it is natural to assume that $Z(x)$ and $Z(x')$ are correlated with covariance

$$Cov(Z(x), Z(x')) = \beta \exp\left[-\sum_{i=1}^{d} \alpha_i (x_i - x'_i)^2\right],$$

where β is the variance of $Z(x)$ while the parameters α_i perform scaling for each dimension.

The best linear model, which we denote $\bar{Y}^n(x)$, of our surface $\mu(x)$, is given by

$$\begin{aligned}\bar{Y}^n(x) &= \sum_{f\in\mathcal{F}} \theta^n_f \phi_f(x) + \\ &\quad \sum_{i=1}^{n} Cov(Z(x_i), Z(x)) \sum_{j=1}^{n} Cov(Z(x_j), Z(x))(\hat{y}_i - \sum_{f\in\mathcal{F}} \theta^n_f \phi_f(x)),\end{aligned}$$

where θ^n is the least squares estimator of the regression parameters, given the n observations $\hat{y}^1, \ldots, \hat{y}^n$.

Kriging starts with the expected improvement in equation (7.45), with a heuristic modification to handle the uncertainty in an experiment (ignored in (7.45)). This gives an adjusted EI of

$$\mathbb{E}^n I(x) = \mathbb{E}^n \left[\max(\bar{Y}^n(x^{**}) - \mu(x), 0)\right] \left(1 - \frac{\sigma_\varepsilon}{\sqrt{\sigma^{2,n}(x) + \sigma^2_\varepsilon}}\right), \qquad (7.46)$$

where x^{**} is a point chosen to maximize a utility that might be given by

$$u^n(x) = -(\bar{Y}^n(x) + \sigma^n(x)).$$

Since x is continuous, maximizing $u^n(x)$ over x can be hard, so we typically limit our search to previously observed points

$$x^{**} = \arg \max_{x\in\{x^1,\ldots,x^n\}} u^n(x).$$

The expectation in (7.46) can be calculated analytically using

$$\begin{aligned}\mathbb{E}^n\left[\max(\bar{Y}^n(x^{**})-\mu(x),0)\right] &= (\bar{Y}^n(x^{**})-\bar{Y}^n(x))\Phi\left(\frac{\bar{Y}^n(x^{**})-\bar{Y}^n(x)}{\sigma^n(x)}\right)\\ &\quad +\sigma^n(x)\phi\left(\frac{\bar{Y}^n(x^{**})-\bar{Y}^n(x)}{\sigma^n(x)}\right),\end{aligned}$$

where $\phi(z)$ is the standard normal density, and $\Phi(z)$ is the cumulative density function for the normal distribution.

Value of information policies are well-suited to problems where information is expensive, since they focus on running the experiments with the highest value of information. These policies are particularly effective when the value of information is concave, which means that the marginal value of each additional experiment is lower than the previous one. This property is not always true, especially when experiments are noisy, as we discussed above.

A lookahead policy such as the knowledge gradient is able to take advantage of more complex belief models than simple lookup table beliefs. As we show below, beliefs may be correlated, or may even be parametric models. This is because the knowledge gradient for, say, alternative x has to consider the beliefs for all $x' \in \mathcal{X}$. This is in contrast with the other index policies (such as upper confidence bounding) where the value associated with alternative x has nothing to do with the beliefs for the other alternatives.

7.7.3 Restricted multiperiod lookahead

The knowledge gradient, which only looks one step ahead, has been found to be particularly effective when the value of information is concave, as we previously depicted in figure 7.4(a). However, when the value of information follows an S-curve, as in figure 7.4(b), the value of doing a single experiment can be almost zero, and provides no guidance toward identifying the best experiments to run.

A common problem with one-step lookahead policies such as the knowledge gradient arises when experiments are very noisy, which means the value of information from a single experiment is quite low. This problem almost always arises when outcomes are 0/1, such as happens when advertise a product and wait to see if a customer clicks on the ad. Needless to say, no-one would ever make a choice about which alternative is best based on 0/1 outcomes; we would perform multiple experiments and take an average. This is what we call a restricted lookahead, since we restrict our evaluation of the future to using one alternative at a time.

We can formalize this notion of performing repeated experiments. Imagine that instead of doing a single experiment, that we can repeat our evaluation of alternative x n_x times. If we are using a lookup table belief model, this means the updated precision is

$$\beta_x^{n+1}(n_x) = \beta_x^n + n_x\beta^W,$$

where as before, β^W is the precision of a single experiment. Then, we compute the knowledge gradient as given in equation (7.44) (the details are given in section 7.8), but using a precision of $n_x\beta^W$ rather than just β^W.

This leaves the question of deciding how to choose n_x. One way is to use the KG(*) algorithm, where we find the value of n_x that produces the highest *average* value of information. We first compute n_x^* from

$$n_x^* = \arg\max_{n_x>0} \frac{\nu_x(n_x)}{n_x}. \tag{7.47}$$

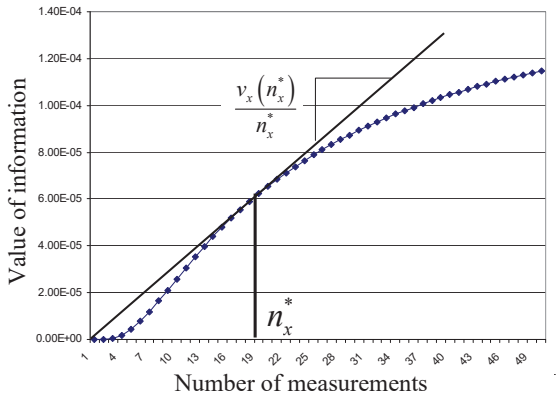


Figure 7.5 The KG(*) policy, which maximizes the average value of a series of experiments testing a single alternative.

This is illustrated in figure 7.5. We do this for each x, and then run the experiment with the highest value of $\frac{v_x(n_x)}{n_x}$. Note that we are not requiring that each experiment be repeated n_x^* times; we only use this to produce a new index (the maximum average value of information), which we use to identify the next single experiment. This is why we call this strategy a *restricted lookahead policy*; we are looking ahead n_x steps, but only considering doing the same experiment x multiple times.

A more general policy uses the concept of *posterior reshaping*. The idea is quite simple. Introduce a repetition parameter θ^{KGLA} where we let the precision of an experiment be given by

$$\beta_x^{n+1}(\theta^{KGLA}) = \beta_x^n + \theta^{KGLA}\beta^W.$$

Now let $\nu_x^{KG,n}(\theta^{KGLA})$ be the knowledge gradient when we use repetition factor θ^{KGLA}. Our knowledge gradient policy would be still given by

$$X^{KG}(S^n|\theta^{KGLA}) = \arg\max_x \nu_x^{KG,n}(\theta^{KGLA}).$$

We now have a tunable parameter, but this is the price of managing this complexity. The value of using a tunable parameter is that the tuning process implicitly captures the effect of the experimental budget N.

7.7.4 Multiperiod deterministic lookahead

Assume we have a setting where experiments are noisy, which means we are likely to face an S-shaped value-of-information curve as depicted in figure 7.4(b). Instead of using θ^{KGLA} as the repeat factor for the knowledge gradient, let y_x be the number of times we plan on repeating the experiment for alternative x given our prior distribution of belief about each alternative. Then let $\nu_x^{KG,0}(y_x)$ be the value-of-information curve for alternative x, using our prior (time 0) belief, if we plan on running the experiment y_x times.

Assume we start with a budget of R^0 experiments. Previously, we assumed we had a budget of N experiments, but this notation will give us a bit more flexibility.

We can determine the vector $y = (y_x)_{x\in\mathcal{X}}$ by solving the optimization problem

$$\max_y \sum_{x\in\mathcal{X}} \nu_x^{KG,0}(y_x), \tag{7.48}$$

subject to the constraints:

$$\sum_{x\in\mathcal{X}} y_x \leq R^0, \tag{7.49}$$

$$y_x \geq 0, \ x \in \mathcal{X}. \tag{7.50}$$

The optimization problem described by equations (7.48)-(7.50) is a non-concave integer programming problem. The good news is that it is very easy to solve optimally using a simple dynamic programming recursion.

Assume that $\mathcal{X} = \{1, 2, ..., M\}$, so that x is an integer between 1 and M. We are going to solve a dynamic program over the alternatives, starting with $x = M$. Let R_x^0 be the number of experiments that we have remaining to allocate over alternatives $x, x+1, \ldots, M$. We start with the last alternative where we need to solve

$$\max_{y_M \leq R_M^0} \nu_M^{KG,0}(y_M). \tag{7.51}$$

Since $\nu_M^{KG,0}(y_M)$ is strictly increasing, the optimal solution would be $y_M = R_M^0$. Now let

$$V_M(R_M) = \nu_M^{KG,0}(R_M),$$

which you obtain for $R_M = 0, 1, \ldots, R^0$ by solving equation (7.51) for each value of R_M^0 (note that you do not have to really "solve" (7.51) at this point, since the solution is just $\nu_M^{KG,0}(y_M)$).

Now that we have $V_M(R_M)$, we step backward through the alternatives using Bellman's recursion:

$$\begin{aligned} V_x(R_x^0) &= \max_{y_x \leq R_x^0} \left(\nu_x^{KG,0}(y_x) + V_{x+1}(R_{x+1}^0)\right) \\ &= \max_{y_x \leq R_x^0} \left(\nu_x^{KG,0}(y_x) + V_{x+1}(R_x^0 - y_x)\right), \end{aligned} \tag{7.52}$$

where equation (7.53) has to be solved for $R_x^0 = 0, 1, \ldots, R^0$. This equation is solved for $x = M-1, M-2, \ldots, 1$.

After we obtain $V_x(R_x^0)$ for each $x \in \mathcal{X}$ and all $0 \leq R_x^0 \leq R^0$, we can then find an optimal allocation y^0 from

$$y_x^0 = \arg\max_{y_x \leq R_x^0} \left(\nu_x^{KG,0}(y_x) + V_{x+1}(R_x^0 - y_x)\right). \tag{7.53}$$

Given the allocation vector $y^0 = (y_x^0)_{x\in\mathcal{X}}$, we now have to decide how to implement this solution. If we can only do one experiment at a time, a reasonable strategy might be to choose the experiment x for which y_x is largest. This would give us a policy that we can write as

$$X^{DLA,n}(S^n) = \arg\max_{x\in\mathcal{X}} y_x^n, \tag{7.54}$$

where we replace y^0 with y^n for iteration n in the calculations above. At iteration n, we would replace R^0 with R^n. After implementing the decision to perform experiment $x^n = X^{DLA,n}(S^n)$, we update $R^{n+1} = R^n - 1$ (assuming we are only performing one experiment at a time). We then observe W^{n+1}, and update the beliefs using our transition function $S^{n+1} = S^M(S^n, x^n, W^{n+1})$ using equations (7.42)-(7.43).

7.7.5 Multiperiod stochastic lookahead policies

A full multiperiod lookahead policy considers making different decisions as we step into the future. We illustrate a full multiperiod lookahead policy for learning using the setting of trying to identify the best hitter on a baseball team. The only way to collect information is to put the hitter into the lineup and observe what happens. We have an estimate of the probability that the player will get a hit, but we are going to update this estimate as we make observations (this is the essence of learning).

Assume that we have three candidates for the position. The information we have on each hitter from previous games is given in Table 7.3. If we choose player A, we have to balance the likelihood of getting a hit, and the value of the information we gain about his true hitting ability, since we will use the event of whether or not he gets a hit to update our assessment of his probability of getting a hit. We are going to again use Bayes' theorem to update our belief about the probability of getting a hit. Fortunately, this model produces some very intuitive updating equations. Let H^n be the number of hits a player has made in n at-bats. Let $\hat{H}^{n+1} = 1$ if a hitter gets a hit in his $(n+1)$st at-bat. Our prior probability of getting a hit after n at-bats is

$$\mathbb{P}[\hat{H}^{n+1} = 1 | H^n, n] = \frac{H^n}{n}.$$

Once we observe $\hat{H}^{n+1}$, it is possible to show that the posterior probability is

$$\mathbb{P}[\hat{H}^{n+2} = 1 | H^n, n, \hat{H}^{n+1}] = \frac{H^n + \hat{H}^{n+1}}{n+1}.$$

In other words, all we are doing is computing the batting average (hits over at-bats).

Our challenge is to determine whether we should try player A, B or C right now. At the moment, A has the best batting average of 0.360, based on a history of 36 hits out of 100 at-bats. Why would we try player B, whose average is only 0.333? We easily see that this statistic is based on only three at-bats, which would suggest that we have a lot of uncertainty in this average.

We can study this formally by setting up the decision tree shown in Figure 7.6. For practical reasons, we can only study a problem that spans two at-bats. We show the current prior probability of a hit, or no hit, in the first at-bat. For the second at-bat, we show only the probability of getting a hit, to keep the figure from becoming too cluttered.

Figure 7.7 shows the calculations as we roll back the tree. Figure 7.7(c) shows the expected value of playing each hitter for exactly one more at-bat using the information obtained from our first decision. It is important to emphasize that after the first decision, only one hitter has had an at-bat, so the batting averages only change for that hitter. Figure 7.7(b) reflects our ability to choose what we think is the best hitter, and Figure 7.7(a) shows

Player	No. hits	No. at-bats	Average
A	36	100	0.360
B	1	3	0.333
C	7	22	0.318

Table 7.3 History of hitting performance for three candidates.

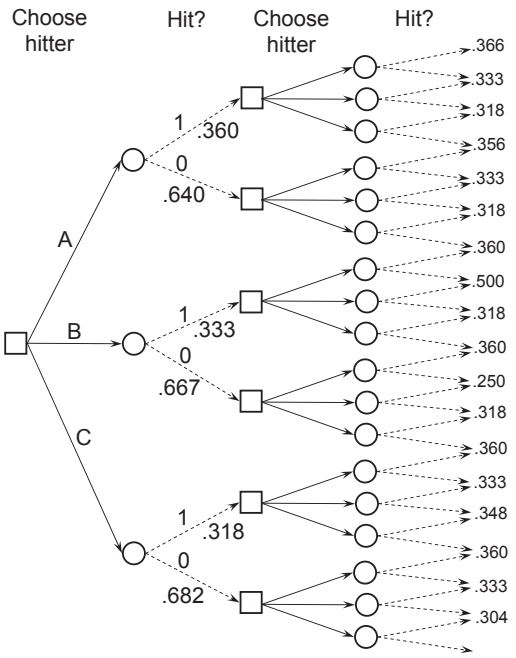


Figure 7.6 The decision tree for finding the best hitter.

the expected value of each hitter before any at-bats have occurred. We use as our reward function the expected number of total hits over the two at-bats. Let R_x be our reward if batter x is allowed to hit, and let H_{1x} and H_{2x} be the number of hits that batter x gets over his two at-bats. Then

$$R_x = H_{1x} + H_{2x}.$$

Taking expectations gives us

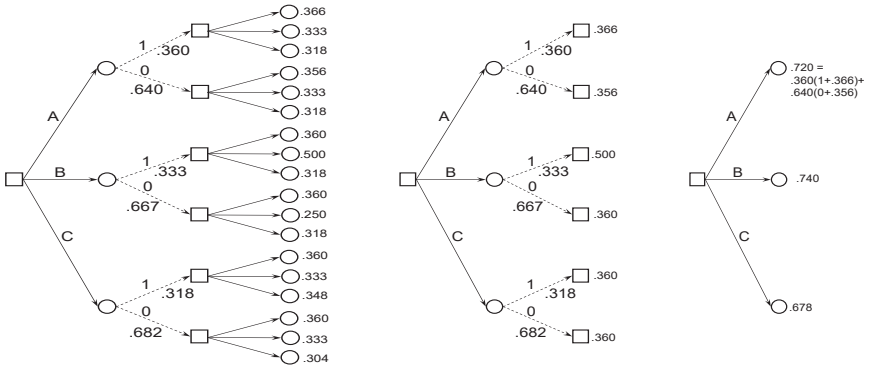
$$\mathbb{E}R_x = \mathbb{E}H_{1x} + \mathbb{E}H_{2x}$$

So, if we choose batter A, the expected number of hits is

$$\begin{aligned} \mathbb{E}R_A &= .360(1 + .366) + .640(0 + .356) \\ &= .720 \end{aligned}$$

where 0.360 is our prior belief about his probability of getting a hit; .366 is the expected number of hits in his second at-bat (the same as the probability of getting a hit) given that he got a hit in his first at-bat. If player A did not get a hit in his first at-bat, his updated probability of getting a hit, 0.356, is still higher than any other player. This means that if we have only one more at-bat, we would still pick player A even if he did not get a hit in his first at-bat.

Although player A initially has the highest batting average, our analysis says that we should try player B for the first at-bat. Why is this? On further examination, we realize that it has a lot to do with the fact that player B has had only three at-bats. If this player



7.7(a) 7.7(b) 7.7(c)

Figure 7.7 (a) Expected value of a hit in the second at-bat; (b) Value of best hitter after one at-bat; (c) Expected value of each hitter before first at-bat.

gets a hit, our estimate of his probability of getting a hit jumps to 0.500, although it drops to 0.250 if he does not get a hit. If player A gets a hit, his batting average moves from 0.360 to 0.366, reflecting the weight of his much longer record. This is our first hint that it can be useful to collect information about choices where there is the greatest uncertainty.

This example illustrates a setting where observations change our beliefs, which we build into the tree. We could have built our tree where all probabilities remain static, which is typical in decision trees. Imbedding the process of updating probabilities within the decision tree is what distinguishes classical decision trees from the use of decision trees in a learning setting.

Decision trees are actually a powerful strategy for learning, although they have not attracted much attention in the learning literature. One reason is simply that they are computationally more difficult, and for most applications, they do not actually work better. Another is that they are harder to analyze, which makes them less interesting in the research communities that analyze algorithms.

7.7.6 Hybrid direct lookahead

When we first introduced direct lookahead policies, we described a strategy of doing a full lookahead using an approximate model. That is, if

$$S^0, x^0, W^1, \ldots, S^n, x^n, W^{n+1}, \ldots$$

represents our base model, the lookahead model might use approximate states $\tilde{S}^{n,m}$, simplified decisions $\tilde{x}^{n,m}$ and/or a sampled information process $\widetilde{W}^{n,m}$.

In addition, we might use a simplified policy $\tilde{X}^{\tilde{\pi}}(\tilde{S}^{n,m})$, as illustrated in the lookahead policy which we first presented in equation (11.24), but which we replicate here:

$$X^{DLA,n}(S^n) = \arg\max_{x^n} \Bigg(C(S^n, x^n) + \tilde{E}\left\{ \max_{\tilde{\pi}} \tilde{E}\left\{ \sum_{m=n+1}^{N} C(\tilde{S}^{n,m}, \tilde{X}^{\tilde{\pi}}(\tilde{S}^{n,m})) | \tilde{S}^{n,n+1} \right\} | S^n, x^n \right\} \Bigg). \tag{7.55}$$

To illustrate a type of hybrid policy, we are not going to approximate the model in any way (the state variables, decisions and observations will all be just as they are in the base model). But we are going to suggest using a simple UCB or interval estimation policy for $\tilde{X}^{\tilde{\pi}}(\tilde{S}^{n,m})$. For example, we might use

$$\tilde{X}^{\tilde{\pi}}(\tilde{S}^{n,m}|\tilde{\theta}^{IE}) = \arg\max_{x\in\mathcal{X}} \left(\tilde{\mu}^{n,m} + \tilde{\theta}^{IE}\tilde{\sigma}^{n,m}\right).$$

Let's see how this works. First, the imbedded optimization over policies in equation (20.24) is replaced with a search over $\tilde{\theta}^{IE}$, which we write as

$$X^{DLA,n}(S^n) = \arg\max_{x^n}\Bigg(C(S^n,x^n) + \tilde{E}\left\{\max_{\tilde{\theta}^{IE}}\tilde{E}\left\{\sum_{m=n+1}^{N} C(\tilde{S}^{n,m},\tilde{X}^{\tilde{\pi}}(\tilde{S}^{n,m}|\tilde{\theta}^{IE}))|\tilde{S}^{n,n+1}\right\}|S^n,x^n\right\}\Bigg). \quad (7.56)$$

This seems easier, but we have to understand what is meant by that $\max$ operator imbedded in the policy. What is happening is that as we make a decision x^n for observation n, we then sample an outcome W^{n+1} given x^n which brings us to a (stochastic) state $\tilde{S}^{n,n+1}$. It is only then that we are supposed to optimize over $\tilde{\theta}^{IE}$, which means we are actually trying to find a function $\tilde{\theta}^{IE}(\tilde{S}_{n,n+1})$. Wow!

Clearly, no-one is going to do this, so we are just going to fix a single parameter θ^{IE}. Note that we no longer have a tilde over it, because it is now a part of the base policy, not the lookahead policy. This means that it is tuned as part of the base policy, so the lookahead is now written

$$X^{DLA,n}(S^n|\theta^{IE}) = \arg\max_{x^n}\Bigg(C(S^n,x^n) + \tilde{E}\left\{\tilde{E}\left\{\sum_{m=n+1}^{N} C(\tilde{S}^{n,m},\tilde{X}^{\tilde{\pi}}(\tilde{S}^{n,m}|\tilde{\theta}^{IE}))|\tilde{S}^{n,n+1}\right\}|S^n,x^n\right\}\Bigg). \quad (7.57)$$

Now we have gotten rid of the imbedded $\max$ operator entirely. We now have a parameterized policy $X^{DLA,n}(S^n|\theta^{IE})$ where θ^{IE} has to be tuned. However, this seems much more manageable.

So, how does our policy $X^{DLA,n}(S^n|\theta^{IE})$ actually work? Assume that our decision x^n is a scalar that we can enumerate. What we can do is for each value of x^n, we can simulate our interval estimation lookahead policy some number of times, and take an average.

We mention this idea primarily to illustrate how we can use a simpler policy inside a lookahead model. Of course, there has to be a reason why we are using a lookahead policy in the first place, so why would we expect a simple IE policy to work? Part of the reason is that approximations within a lookahead policy do not introduce the same errors as if we tried to use this policy instead of the lookahead policy.

7.8 THE KNOWLEDGE GRADIENT (CONTINUED)*

The knowledge gradient belongs to the class of value-of-information policies which choose alternatives based on the improvement in the quality of the objective from better decisions that arise from a better understanding of the problem. The knowledge gradient works from a Bayesian belief model where our belief about the truth is represented by a probability

distribution of possible truths. The basic knowledge gradient calculates the value of a single experiment, but this can be used as a foundation for variations that allow for repeated experiments.

The knowledge gradient was originally developed for offline (final reward) settings, so we begin with this problem class. Our experience is that the knowledge gradient is particularly well suited for settings where experiments (or observations) are expensive. For example:

- An airline wants to know the effect of allowing additional schedule slack, which can only be evaluated by running dozens of simulations to capture the variability due to weather. Each simulation may take several hours to run.
- A scientist needs to evaluate the effect of increasing the temperature of a chemical reaction or the strength of a material. A single experiment may take several hours, and needs to be repeated to reduce the effect of the noise in each experiment.
- A drug company is running clinical trials on a new drug, where it is necessary to test the drug at different dosages for toxicity. It takes several days to assess the effect of the drug at a particular dosage.

After developing the knowledge gradient for offline (final reward) settings, we show how to compute the knowledge gradient for online (cumulative reward) problems. We begin by discussing belief models, but devote the rest of this section to handling the special case of independent beliefs. Section 7.8.4 extends the knowledge gradient to a general class of nonlinear parametric belief models.

7.8.1 The belief model

The knowledge gradient uses a Bayesian belief model where we begin with a prior on $\mu_x = \mathbb{E}F(x, W)$ for $x \in \{x_1, \ldots, x_M\}$. We are going to illustrate the key ideas using a lookup table belief model (which is to say, we have an estimate for each value of x), where we initially assume the beliefs are independent. This means that anything we learn about some alternative x does not teach us anything about an alternative x'.

We assume that we believe that the true value of μ_x is described by a normal distribution $N(\bar{\mu}_x^0, \bar{\sigma}_x^{2,0})$, known as the prior. This may be based on prior experience (such as past experience with the revenue from charging a price x for a new book), some initial data, or from an understanding of the physics of a problem (such as the effect of temperature on the conductivity of a metal).

It is possible to extend the knowledge gradient to a variety of belief models. A brief overview of these is:

Correlated beliefs Alternatives x may be related, perhaps because they are discretizations of a continuous parameter (such as temperature or price), so that μ_x and μ_{x+1} are close to each other. Trying x then teaches us something about μ_{x+1}. Alternatively, x and x' may be two drugs in the same class, or a product with slightly different features. We capture these relationships with a covariance matrix Σ^0 where $\Sigma^0_{xx'} = Cov(\mu_x, \mu_{x'})$. We show how to handle correlated beliefs below.

Parametric linear models We may derive a series of features $\phi_f(x)$, for $f \in \mathcal{F}$. Assume that we represent our belief using

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

where $f(x|\theta) \approx \mathbb{E}F(x,W)$ is our estimate of $\mathbb{E}F(x,W)$. We now treat θ as the unknown parameter, where we might assume that the vector θ is described by a multivariate normal distribution $N(\theta^0, \Sigma^{\theta,0})$, although coming up with these priors (in the parameter space) can be tricky.

Parametric nonlinear models Our belief model might be nonlinear in θ. For example, we might use a logistic regression

$$f(x|\theta) = \frac{e^{U(x|\theta)}}{1+e^{U(x|\theta)}}, \tag{7.58}$$

where $U(x|\theta)$ is a linear model given by

$$U(x|\theta) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_K x_K$$

where $(x_1, \ldots, x_K)$ are the features of a decision x.

Belief models that are nonlinear in the parameters can cause some difficulty, but we can circumvent this by using a sampled belief model, where we assume the uncertain θ is one of the set $\{\theta_1, \ldots, \theta_K\}$. Let p_k^n be the probability that $\theta = \theta_k$, which means that $p^n = (p_k^n),\ k = 1, \ldots, K$ is our belief at time n. See section 3.9.2 for more information.

Nonparametric models Simpler nonparametric models are primarily local approximations, so we could use constant, linear or nonlinear models defined over local regions. More advanced models include neural networks (the kind known as "deep learners") or support vector machines, both of which were introduced in chapter 3.

Below we show how to calculate the knowledge gradient for each of these belief models, with the exception of the nonparametric models (listed for completeness).

7.8.2 The knowledge gradient for maximizing final reward

The knowledge gradient seeks to learn about the value of different actions by maximizing the value of information from a single observation. Let S^n be our belief state about the value of each action x. The knowledge gradient uses a Bayesian model, so

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

captures the mean and variance of our belief about the true value $\mu_x = \mathbb{E}F(x,W)$, where we also assume that $\mu_x \sim N(\bar{\mu}_x^n, \sigma_x^{2,n})$.

The value of being in belief state S^n is given by

$$V^n(S^n) = \mu_{x^n},$$

where x^n is the choice that appears to be best given what we know after n experiments, calculated using

$$x^n = \arg\max_{x' \in \mathcal{X}} \bar{\mu}_{x'}^n.$$

If we choose action x^n, we then observe $W_{x^n}^{n+1}$ which we then use to update our estimate of our belief about μ_x using our Bayesian updating equations (7.42)-(7.43).

The value of state $S^{n+1}(x)$ when we try action x is given by

$$V^{n+1}(S^{n+1}(x)) = \max_{x' \in \mathcal{X}} \bar{\mu}_{x'}^{n+1}(x).$$

where $\bar{\mu}_{x'}^{n+1}(x)$ is the updated estimate of $\mathbb{E}\mu$ given S^n (that is, our estimate of the distribution of μ after n experiments), and the result of implementing x and observing W_x^{n+1}. We have to decide which experiment to run after the n^{th} observation, so we have to work with the expected value of running experiment x, given by

$$\mathbb{E}\{V^{n+1}(S^{n+1}(x))|S^n\} = \mathbb{E}\{\max_{x' \in \mathcal{X}} \bar{\mu}_{x'}^{n+1}(x)|S^n\}.$$

The knowledge gradient is then given by

$$\nu_x^{KG,n} = \mathbb{E}\{V^{n+1}(S^M(S^n, x, W^{n+1}))|S^n, x\} - V^n(S^n),$$

which is equivalent to

$$\nu^{KG}(x) = \mathbb{E}\{\max_{x'} \bar{\mu}_{x'}^{n+1}(x)|S^n\} - \max_{x'} \bar{\mu}_{x'}^n. \tag{7.59}$$

Here, $\bar{\mu}^{n+1}(x)$ is the updated value of $\bar{\mu}^n$ after running an experiment with setting $x = x^n$, after which we observe W_x^{n+1}. Since we have not yet run the experiment, W_x^{n+1} is a random variable, which means that $\bar{\mu}^{n+1}(x)$ is random. In fact, $\bar{\mu}^{n+1}(x)$ is random for two reasons. To see this, we note that when we run experiment x, we observe an updated value from

$$W_x^{n+1} = \mu_x + \varepsilon_x^{n+1},$$

where $\mu_x = \mathbb{E}F(x, W)$ is the true value, while ε_x^{n+1} is the noise in the observation. This introduces two forms of uncertainty: the unknown truth μ_x, and the noise ε_x^{n+1}. Thus, it would be more accurate to write equation (7.59) as

$$\nu^{KG}(x) = \mathbb{E}_\mu\{\mathbb{E}_{W|\mu} \max_{x'} \bar{\mu}_{x'}^{n+1}(x)|S^n\} - \max_{x'} \bar{\mu}_{x'}^n. \tag{7.60}$$

where the first expectation $\mathbb{E}_\mu$ is conditioned on our belief state S^n, while the second expectation $\mathbb{E}_{W|\mu}$ is over the experimental noise W given our distribution of belief about the truth μ.

To illustrate how equation (7.60) is calculated, imagine that μ takes on values $\{\mu_1, \ldots, \mu_K\}$, and that p_k^μ is the probability that $\mu = \mu_k$. Assume that μ is the mean of a Poisson distribution describing the number of customers W that click on a website and assume that

$$P^W[W = \ell|\mu = \mu_k] = \frac{\mu_k^\ell e^{-\mu_k}}{\ell!}.$$

We would then compute the expectation in equation (7.60) using

$$\nu^{KG}(x) = \sum_{k=1}^K \left(\sum_{\ell=0}^\infty \left(\max_{x'} \bar{\mu}_{x'}^{n+1}(x|W = \ell) \right) P^W[W = \ell|\mu = \mu_k] \right) p_k^\mu - \max_{x'} \bar{\mu}_{x'}^n.$$

where $\bar{\mu}_{x'}^{n+1}(x|W = \ell)$ is the updated estimate of $\bar{\mu}_{x'}^n$ if we run experiment x (which might be a price or design of a website) and we then observe $W = \ell$. The updating would be done using any of the recursive updating equations described in chapter 3.

We now want to capture how well we can solve our optimization problem, which means solving $\max_{x'} \bar{\mu}_{x'}^{n+1}(x)$. Since $\bar{\mu}_{x'}^{n+1}(x)$ is random (since we have to pick x before we know W^{n+1}), then $\max_{x'} \bar{\mu}_{x'}^{n+1}(x)$ is random. This is why we have to take the expectation, which is conditioned on S^n which captures what we know now.

Computing a knowledge gradient policy for independent beliefs is extremely easy. We assume that all rewards are normally distributed, and that we start with an initial estimate of the mean and variance of the value of decision x, given by

$$\begin{aligned}\bar{\mu}_x^0 &= \text{The initial estimate of the expected reward from making decision } x,\\ \bar{\sigma}_x^0 &= \text{The initial estimate of the standard deviation of our belief about } \mu.\end{aligned}$$

Each time we make a decision we receive a reward given by

$$W_x^{n+1} = \mu_x + \varepsilon^{n+1},$$

where μ_x is the true expected reward from action x (which is unknown) and ε is the experimental error with standard deviation σ_W (which we assume is known).

The estimates $(\bar{\mu}_x^n, \bar{\sigma}_x^{2,n})$ are the mean and variance of our belief about μ_x after n observations. We are going to find that it is more convenient to use the idea of *precision* (as we did in chapter 3) which is the inverse of the variance. So, we define the precision of our belief and the precision of the experimental noise as

$$\begin{aligned}\beta_x^n &= 1/\bar{\sigma}_x^{2,n},\\ \beta^W &= 1/\sigma_W^2.\end{aligned}$$

If we take action x and observe a reward W_x^{n+1}, we can use Bayesian updating to obtain new estimates of the mean and variance for action x, following the steps we first introduced in section 3.4. To illustrate, imagine that we try an action x where $\beta_x^n = 1/(20^2) = 0.0025$, and $\beta^W = 1/(40^2) = .000625$. Assume $\bar{\mu}_x^n = 200$ and that we observe $W_x^{n+1} = 250$. The updated mean and precision are given by

$$\begin{aligned}\bar{\mu}_x^{n+1} &= \frac{\beta_x^n \bar{\mu}_x^n + \beta^W W_x^{n+1}}{\beta_x^n + \beta^W}\\ &= \frac{(.0025)(200) + (.000625)(250)}{.0025 + .000625}\\ &= 210.\\ \beta_x^{n+1} &= \beta_x^n + \beta^W\\ &= .0025 + .000625\\ &= .003125.\end{aligned}$$

We next find the variance of the *change* in our estimate of μ_x assuming we choose to sample action x in iteration n. For this we define

$$\begin{aligned}\tilde{\sigma}_x^{2,n} &= \mathit{Var}[\bar{\mu}_x^{n+1} - \bar{\mu}_x^n | S^n] &(7.61)\\ &= \mathit{Var}[\bar{\mu}_x^{n+1} | S^n]. &(7.62)\end{aligned}$$

We use the form of equation (7.61) to highlight the definition of $\tilde{\sigma}_x^{2,n}$ as the change in the variance given what we know at time n, but when we condition on what we know (captured by S^n) it means that $\mathit{Var}[\bar{\mu}_x^n | S^n] = 0$ since $\bar{\mu}_x^n$ is just a number at time n.

With a little work, we can write $\tilde{\sigma}_x^{2,n}$ in different ways, including

$$\begin{aligned}\tilde{\sigma}_x^{2,n} &= \bar{\sigma}_x^{2,n} - \bar{\sigma}_x^{2,n+1}, &(7.63)\\ &= \frac{(\bar{\sigma}_x^{2,n})}{1+\sigma_W^2/\bar{\sigma}_x^{2,n}}. &(7.64)\end{aligned}$$

Equation (7.63) expresses the (perhaps unexpected) result that $\tilde{\sigma}_x^{2,n}$ measures the change in the estimate of the standard deviation of the reward from decision x from iteration $n-1$ to n. Using our numerical example, equations (7.63) and (7.64) both produce the result

$$\begin{aligned}\tilde{\sigma}_x^{2,n} &= 400 - 320 = 80\\ &= \frac{40^2}{1+\frac{10^2}{40^2}} = 80.\end{aligned}$$

Finally, we compute

$$\zeta_x^n = -\left|\frac{\bar{\mu}_x^n - \max_{x'\neq x}\bar{\mu}_{x'}^n}{\tilde{\sigma}_x^n}\right|.$$

ζ_x^n is called the *normalized influence* of decision x. It gives the number of standard deviations from the current estimate of the value of decision x, given by $\bar{\mu}_x^n$, and the best alternative other than decision x. We then find

$$f(\zeta) = \zeta\Phi(\zeta) + \phi(\zeta),$$

where $\Phi(\zeta)$ and $\phi(\zeta)$ are, respectively, the cumulative standard normal distribution and the standard normal density. Thus, if Z is normally distributed with mean 0, variance 1, $\Phi(\zeta) = \mathbb{P}[Z \leq \zeta]$ while

$$\phi(\zeta) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{\zeta^2}{2}\right).$$

The knowledge gradient algorithm chooses the decision x with the largest value of $\nu_x^{KG,n}$ given by

$$\nu_x^{KG,n} = \tilde{\sigma}_x^n f(\zeta_x^n).$$

The knowledge gradient algorithm is quite simple to implement. Table 7.4 illustrates a set of calculations for a problem with five options. $\bar{\mu}$ represents the current estimate of the value of each action, while $\bar{\sigma}$ is the current standard deviation of μ. Options 1, 2 and 3 have the same value for $\bar{\sigma}$, but with increasing values of $\bar{\mu}$.

The table illustrates that when the variance is the same, the knowledge gradient prefers the decisions that appear to be the best. Decisions 3 and 4 have the same value of $\bar{\mu}$, but decreasing values of $\bar{\sigma}$, illustrating that the knowledge gradient prefers decisions with the highest variance. Finally, decision 5 appears to be the best of all the decisions, but has the lowest variance (meaning that we have the highest confidence in this decision). The knowledge gradient is the smallest for this decision out of all of them.

The knowledge gradient trades off how well an alternative is expected to perform, and how uncertain we are about this estimate. Figure 7.8 illustrates this tradeoff. Figure 7.8(a) shows five alternatives, where the estimates are the same across all three alternatives, but with increasing standard deviations. Holding the mean constant, the knowledge gradient

Decision	$\bar{\mu}$	$\bar{\sigma}$	$\tilde{\sigma}$	ζ	$f(z)$	KG index
1	1.0	2.5	1.569	-1.275	0.048	0.075
2	1.5	2.5	1.569	-0.956	0.090	0.142
3	2.0	2.5	1.569	-0.637	0.159	0.249
4	2.0	2.0	1.400	-0.714	0.139	0.195
5	3.0	1.0	0.981	-1.020	0.080	0.079

Table 7.4 The calculations behind the knowledge gradient algorithm

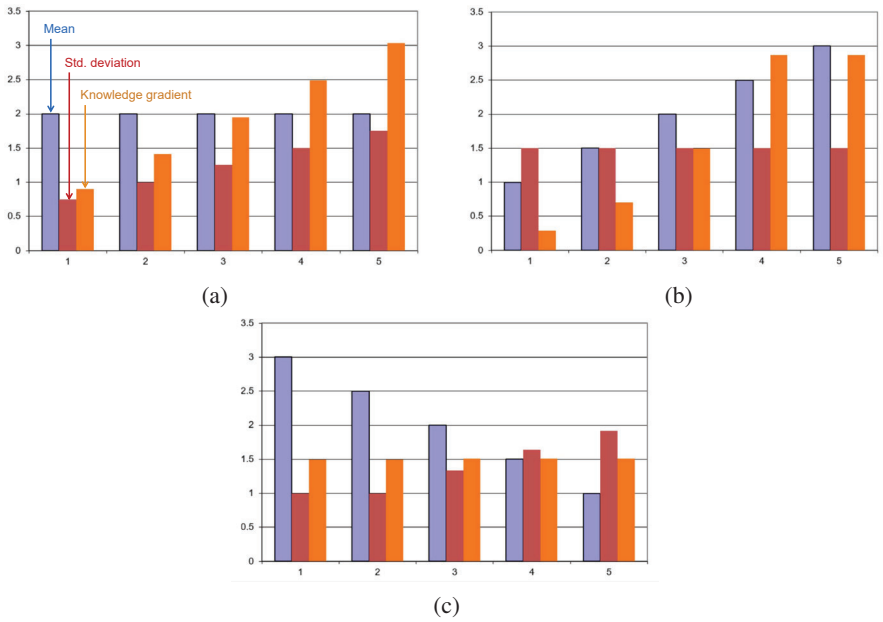


Figure 7.8 The knowledge gradient for lookup table with independent beliefs with equal means (a), equal variances (b), and adjusting means and variances so that the KG is equal (c).

increases with standard deviation of the estimate of the mean. Figure 7.8(b) repeats this exercise, but now holding the standard deviation the same, with increasing means, showing that the knowledge gradient increases with the estimate of the mean. Finally, figure 7.8(c) varies the estimates of the mean and standard deviation so that the knowledge gradient stays constant, illustrating the tradeoff between the estimated mean and its uncertainty.

This tradeoff between the expected performance of a design, and the uncertainty about its performance, is a feature that runs through well designed policies. However, all the other policies with this property (interval estimation, upper confidence bounding, Gittins indices), achieve this with indices that consist of the sum of the expected performance and a term that reflects the uncertainty of an alternative.

The knowledge gradient, however, achieves this behavior without the structure of the expected reward plus an uncertainty term with a tunable parameter. In fact, this brings out a major feature of the knowledge gradient, which is that it does not have a tunable parameter.

7.8.3 The knowledge gradient for maximizing cumulative reward

There are many online applications of dynamic programming where there is an operational system which we would like to optimize in the field. In these settings, we have to live with the rewards from each experiment. As a result, we have to strike a balance between the value of an action and the information we gain that may improve our choice of actions in the future. This is precisely the tradeoff that is made by Gittins indices for the multiarmed bandit problem.

It turns out that the knowledge gradient is easily adapted for online problems. As before, let $\nu_x^{KG,n}$ be the offline knowledge gradient, giving the value of observing action x, measured in terms of the improvement in a single decision. Now imagine that we have a budget of N decisions. After having made n decisions (which means, n observations of the value of different actions), if we observe $x = x^n$ which allows us to observe W_x^{n+1}, we received an expected reward of $\mathbb{E}^n W_x^{n+1} = \bar{\mu}_x^n$, and obtain information that improves the contribution from a single decision by $\nu_x^{KG,n}$. However, we have $N - n$ more decisions to make. Assume that we learn from the observation of W_x^{n+1} by choosing $x^n = x$, but we do not allow ourselves to learn anything from future decisions. This means that the remaining $N - n$ decisions have access to the same information.

From this analysis, the knowledge gradient for online applications consists of the expected value of the single-period contribution of the experiment, plus the improvement in all the remaining decisions in our horizon. This implies

$$\nu_x^{OLKG,n} = \bar{\mu}_x^n + (N - n)\nu_x^{KG,n}. \tag{7.65}$$

This is a general formula that extends any calculation of the offline knowledge gradient to online (cumulative reward) learning problems. Note that we now obtain the same structure we previously saw in UCB and IE policies (as well as Gittins indices) where the index is a sum of a one-period reward, plus a bonus term for learning.

It is also important to recognize that the online policy is time-dependent, because of the presence of the $(N - n)$ coefficient. When n is small, $(N - n)$ is large, and the policy will emphasize exploration. As we progress, $(N - n)$ shrinks and we place more emphasis on maximizing the online reward.

7.8.4 The knowledge gradient for sampled belief model*

There are many settings where our belief model is nonlinear in the parameters. For example, imagine that we are modeling the probability that a customer will click on an ad when we bid x, where higher bids improve our ability of getting attractive placements that increase the number of clicks. Assume that the response is a logistic regression given by

$$P^{purchase}(x|\theta) = \frac{e^{U(x|\theta)}}{1 + e^{U(x|\theta)}} \tag{7.66}$$

where $U(x|\theta) = \theta_0 + \theta_1 x$. We do not know θ, so we are going to represent it as a random variable with some distribution. We might represent it as a multivariate normal distribution, but this makes computing the knowledge gradient very complicated.

A very practical strategy is to use a sampled belief model which we first introduced in section 3.9.2. Using this approach we assume that θ takes an outcome in the set $\{\theta_1, \ldots, \theta_K\}$. Let $p_k^n = Prob[\theta = \theta_k]$ after we have run n experiments. Note that these probabilities represent our belief state, which means that

$$S^n = (p_k^n)_{k=1}^K.$$

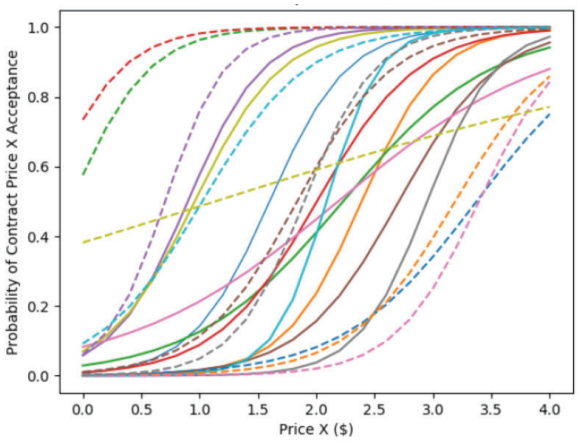


Figure 7.9 Sampled set of bid response curves.

We might use an initial distribution $p_k^0 = 1/K$. A sampled belief for a logistic curve is illustrated in figure 7.9.

Let W_x^{n+1} be the observation when we run experiment $x^n = x$, and let θ^{n+1} be the random variable representing θ after we have run experiment x and observed W_x^{n+1}. When we used a lookup table belief model, we wrote the knowledge gradient as (see (7.59))

$$\nu^{KG,n}(x) = \mathbb{E}\{\max_{x'} \bar{\mu}_{x'}^{n+1}(x)|S^n, x^n = x\} - \max_{x'} \bar{\mu}_{x'}^n. \tag{7.67}$$

For our nonlinear model, we let $\mu_x = f(x|\theta)$ where we assume we know the function $f(x|\theta)$ but we do not know θ. The knowledge gradient would then be written

$$\begin{aligned}\nu^{KG,n}(x) = &\mathbb{E}\{\max_{x'} \mathbb{E}\{f(x', \theta^{n+1}(x))|S^{n+1}\}|S^n, x^n = x\} \\ &- \max_{x'} \mathbb{E}_\theta\{f(x', \theta)|S^n\}.\end{aligned} \tag{7.68}$$

We are going to step through this expression more carefully since we are going to have to compute it directly. Readers just interested in computing the knowledge gradient can jump right to equation (7.76) which can be directly implemented. The derivation that we provide next will provide insights into the principles of the knowledge gradient as a concept.

First, we need to realize that $\bar{\mu}_x^n = \mathbb{E}\{\mu_x|S^n\}$ is the expectation of μ_x given what we know after n iterations, which is captured in S^n. For our nonlinear model, this would be written

$$\begin{aligned}\mathbb{E}_\theta\{f(x', \theta)|S^n\} &= \sum_{k=1}^K p_k^n f(x', \theta_k) \\ &= \bar{f}^n(x').\end{aligned}$$

Next, $\bar{\mu}_{x'}^{n+1}(x)$ in equation (7.67) is our estimate of μ_x after running experiment $x^n = x$ and observing W_x^{n+1}. For the lookup table model, we would write this as

$$\bar{\mu}_{x'}^{n+1}(x) = \mathbb{E}_\mu\{\mu_{x'}|S^{n+1}\} \tag{7.69}$$

where $S^{n+1} = S^M(S^n, x^n, W^{n+1})$. This means that (7.69) can also be written

$$\bar{\mu}_{x'}^{n+1}(x) = \mathbb{E}_\mu\{\mu_{x'}|S^n, x^n, W^{n+1}\} \tag{7.70}$$

For our sampled belief model, we use p_k^n instead of $\bar{\mu}^n$, and we use the updated probabilities $p_k^{n+1}(S^n, x^n = s, W_x^{n+1} = W)$ instead of $\bar{\mu}^{n+1}(x)$ where

$$p_k^{n+1}(S^n, x^n = x, W_x^{n+1} = W) = Prob[\theta = \theta_k|S^n, x^n = x, W_x^{n+1} = W].$$

We express the dependence of $p_k^{n+1}(S^n, x^n = s, W_x^{n+1} = W)$ on the prior state S^n, decision x^n and experimental outcome W^{n+1} explicitly to make these dependencies clear. The random variable $W = W_x^{n+1}$ depends on θ since

$$W_x^{n+1} = f(x|\theta) + \varepsilon^{n+1}.$$

Our belief about θ depends on when we are taking the expectation, which is captured by conditioning on S^n (or later, S^{n+1}). To emphasize the dependence on S^n, we are going to write $\mathbb{E}^n\{\cdot|S^n\}$ to emphasize when we are conditioning on S^n. This will help when we have to use nested expectations, conditioning on both S^n and S^{n+1} in the same equation.

The expectation inside the max operator is

$$\begin{aligned}\mathbb{E}_\theta^{n+1}\{f(x', \theta^{n+1}(x))|S^{n+1}\} &= \mathbb{E}_\theta^{n+1}\{f(x', \theta^{n+1}(x))|S^n, x^n = x, W_x^{n+1} = W\} \\ &= \sum_{k=1}^K f(x', \theta_k)p_k^{n+1}(S^n, x^n = x, W_x^{n+1} = W).\end{aligned}$$

Note that we are only taking the expectation over θ, since W^{n+1} is known at this point. We take the expectation over θ given the posterior p_k^{n+1} because even after we complete the $n+1^{st}$ experiment, we still have to make a decision (that is, choosing x') without knowing the true value of θ.

We now have to compute $p_k^{n+1}(S^n, x^n = x, W_x^{n+1} = W)$. We first assume that we know the distribution of W_x^{n+1} given θ (that is, we know the distribution of W_x^{n+1} *if* we know θ). For our ad-click problem, this would just have outcomes 0 or 1 where $Prob[W = 1]$ is given by our logistic curve in equation (7.66). For more general problems, we are going to assume we have the distribution.

$$f^W(w|x, \theta_k) \quad = \quad \mathbb{P}[W^{n+1} = w|x, \theta = \theta_k].$$

We compute $p_k^{n+1}(S^n, x^n = x, W_x^{n+1} = W)$ using Bayes theorem by first writing

$$\begin{aligned}p_k^{n+1}(S^n, x^n = x, W_x^{n+1} = w) &= Prob[\theta = \theta_k|S^n, x^n = x, W_x^{n+1} = w] \\ &= \frac{Prob[W_x^{n+1} = w|\theta = \theta_k, S^n, x^n = x]Prob[\theta = \theta_k|S^n, x^n = x]}{Prob[W_x^{n+1} = w|S^n, x^n = x]} \\ &= \frac{f^W(W^{n+1} = w|x^n, \theta_k)p_k^n}{C(w)},\end{aligned} \tag{7.71}$$

where $C(w)$ is the normalizing constant given $W^{n+1} = w$, which is calculated using

$$C(w) = \sum_{k=1}^K f^W(W^{n+1} = w|x^n, \theta_k)p_k^n.$$

Below, we are going to treat $C(W)$ (with capital W) as a random variable with realization $C(w)$. Note that we condition $p_k^{n+1}(S^n, x^n = x, W_x^{n+1} = w)$ on S^n since this gives us the prior p_k^n which we use in Bayes theorem. However, once we have computed $p_k^{n+1}(S^n, x^n = x, W_x^{n+1} = w)$ we write the posterior probability as $p_k^{n+1})(w)$ since we no longer need to remember $x^n = x$ or the prior distribution $S^n = (p_k^n)_{k=1}^K$, but we do need to express the dependence on the outcome $W^{n+1} = w$. We will write the posterior distribution as $p^n(W)$ when we want to express the outcome as a random variable.

We are now ready to compute the knowledge gradient in equation (7.68). We begin by writing it with expanded expectations as

$$\nu^{KG,n}(x) = \mathbb{E}_\theta^n \mathbb{E}_{W^{n+1}|\theta}\{\max_{x'} \mathbb{E}_\theta^{n+1}\{f(x', \theta^{n+1})|S^{n+1}\}|S^n, x^n = x\} - \max_{x'} \mathbb{E}_\theta^n\{f(x', \theta)|S^n\}. \tag{7.72}$$

We have to take the expectations $\mathbb{E}_\theta^n \mathbb{E}_{W^{n+1}|\theta}$ because when we are trying to decide which experiment x to run, we do not know the outcome W^{n+1}, and we do not know the true value of θ on which W^{n+1} depends.

The posterior distribution of belief allows us to write $\mathbb{E}_\theta^{n+1} f(x', \theta^{n+1})|S^{n+1}\}$ using

$$\mathbb{E}_\theta^{n+1}\{f(x', \theta^{n+1})|S^{n+1}\} \quad = \quad \sum_{k=1}^K f(x', \theta_k) p_k^{n+1}(W^{n+1}).$$

Substituting this into equation (7.72) gives us

$$\nu^{KG,n}(x) \quad = \quad \mathbb{E}_\theta^n \mathbb{E}_{W^{n+1}|\theta} \left\{ \max_{x'} \sum_{k=1}^K f(x', \theta_k) p_k^{n+1}(W^{n+1}) \middle| S^n, x^n = x \right\} - \max_{x'} \bar{f}^n(x'). \tag{7.73}$$

We now focus on computing the first term of the knowledge gradient. Substituting $p_k^{n+1}(W^{n+1})$ from equation (7.71) into (7.73) gives us

$$\mathbb{E}_\theta^n \mathbb{E}_{W^{n+1}|\theta} \left\{ \max_{x'} \sum_{k=1}^K f(x', \theta_k) p_k^{n+1}(W^{n+1}) \middle| S^n \right\} \quad =$$
$$\mathbb{E}_\theta^n \mathbb{E}_{W^{n+1}|\theta} \left\{ \max_{x'} \sum_{k=1}^K f(x', \theta_k) \left(\frac{f^W(W^{n+1}|x^n, \theta_k) p_k^n}{C(W^{n+1})} \right) |S^n, x = x^n \right\}.$$

Keeping in mind that the entire expression is a function of x, the expectation can be written

$$\begin{aligned}
&\mathbb{E}_\theta^n \mathbb{E}_{W^{n+1}|\theta} \left\{ \max_{x'} \frac{1}{C(W)} \sum_{k=1}^K f(x', \theta) \big(f^W(W^{n+1}|x^n, \theta_k) p_k^n \big) |S^n, x = x^n \right\} \\
&= \quad \mathbb{E}_\theta^n \mathbb{E}_{W|\theta} \frac{1}{C(W)} \left\{ \max_{x'} \sum_{k=1}^K f(x', \theta) \big(f^W(W^{n+1}|x^n, \theta_k) p_k^n \big) |S^n, x = x^n \right\} \\
&= \quad \sum_{j=1}^K \left(\sum_{\ell=1}^L \frac{1}{C(w_\ell)} \{A_\ell\} f^W(W^{n+1} = w_\ell | x, \theta_j) \right) p_j^n. \qquad (7.74)
\end{aligned}$$

where

$$A_\ell = \max_{x'} \sum_{k=1}^{K} f(x', \theta_k)\big(f^W(W^{n+1} = w_\ell | x^n, \theta_k) p_k^n\big).$$

We pause to note that the density $f^W(w, x, \theta)$ appears twice in equation (7.74): once as $f^W(W^{n+1} = w_\ell | x^n, \theta_k)$, and once as $f^W(W^{n+1} = w_\ell | x, \theta_j)$. The first one entered the equation as part of the use of Bayes' theorem to find $p_x^{n+1}(W)$. This calculation is done inside the max operator after W^{n+1} has been observed. The second one arises because when we are deciding the experiment x^n, we do not yet know W^{n+1} and we have to take the expectation over all possible outcomes. Note that if we have binary outcomes (1 if the customer clicks on the ad, 0 otherwise), then the summation over w_ℓ is only over those two values.

We can further simplify this expression by noticing that the terms $f^W(W = w_\ell | x, \theta_j)$ and p_j^n are not a function of x' or k, which means we can take them outside of the $\max$ operator. We can then reverse the order of the other sums over k and w_ℓ, giving us

$$\begin{aligned}
&\mathbb{E}_\theta \mathbb{E}_{W|\theta} \left\{ \max_{x'} \frac{1}{C(W)} \sum_{k=1}^{K} f(x', \theta_k f^W(W | x^n, \theta_k)) p_k^n | S^n, x = x^n \right\} \\
&= \sum_{\ell=1}^{L} \sum_{j=1}^{K} \left(\frac{f^W(W = w_\ell | x, \theta_j) p_j^n}{C(w_\ell)} \right) \left\{ \max_{x'} \sum_{k=1}^{K} f(x', \theta_k) f^W(W = w_\ell | x^n, \theta_k) p_k^n | S^n, x = x^n \right\}.
\end{aligned} \tag{7.75}$$

Using the definition of the normalizing constant $C(w)$ we can write

$$\begin{aligned}
\sum_{j=1}^{K} \left(\frac{f^W(W = w_\ell | x, \theta_j) p_j^n}{C(w_\ell)} \right) &= \left(\frac{\sum_{j=1}^{K} f^W(W = w_\ell | x, \theta_j) p_j^n}{C(w_\ell)} \right) \\
&= \left(\frac{\sum_{j=1}^{K} f^W(W = w_\ell | x, \theta_j) p_j^n}{\sum_{k=1}^{K} f^W(W = w_\ell | x, \theta_k) p_k^n} \right) \\
&= 1.
\end{aligned}$$

We just simplified the problem by cancelling two summations over the K values of θ. This is a significant simplification, since these sums were nested. This allows us to write (7.75) as

$$\begin{aligned}
&\mathbb{E}_\theta \mathbb{E}_{W|\theta} \left\{ \max_{x'} \frac{1}{C(W)} \sum_{k=1}^{K} p_k^n f^W(W | x^n, \theta_k) f(x', \theta_k) | S^n, x = x^n \right\} \\
&\quad = \sum_{\ell=1}^{L} \left\{ \max_{x'} \sum_{k=1}^{K} p_k^n f^W(W = w_\ell | x^n, \theta_k) f(x', \theta_k) | S^n, x = x^n \right\}.
\end{aligned} \tag{7.76}$$

This is surprisingly powerful logic, since it works with any nonlinear belief model.

7.8.5 Knowledge gradient for correlated beliefs

A particularly important feature of the knowledge gradient is that it can be adapted to handle the important problem of correlated beliefs. In fact, the vast majority of real applications exhibit some form of correlated beliefs. Some examples are given below.

EXAMPLE 7.1

Correlated beliefs can arise when we are maximizing a continuous surface (nearby points will be correlated) or choosing subsets (such as the location of a set of facilities) which produce correlations when subsets share common elements. If we are trying to estimate a continuous function, we might assume that the covariance matrix satisfies

$$Cov(x, x') \propto e^{-\rho\|x-x'\|},$$

where ρ captures the relationship between neighboring points. If x is a vector of $0's$ and $1's$ indicating elements in a subset, the covariance might be proportional to the number of 1's that are in common between two choices.

EXAMPLE 7.2

There are about two dozen drugs for reducing blood sugar, divided among four major classes. Trying a drug in one class can provide an indication of how a patient will respond to other drugs in that class.

EXAMPLE 7.3

A materials scientist is testing different catalysts in a process to design a material with maximum conductivity. Prior to running any experiment, the scientist is able to estimate the likely relationship in the performance of different catalysts, shown in table 7.5. The catalysts that share an Fe (iron) or Ni (nickel) molecule show higher correlations.

	1.4nmFe	1nmFe	2nmFe	10nm-Fe	2nmNi	Ni0.6nm	10nm-Ni
1.4nmFe	1.0	0.7	0.7	0.6	0.4	0.4	0.2
1nmFe	0.7	1.0	0.7	0.6	0.4	0.4	0.2
2nmFe	0.7	0.7	1.0	0.6	0.4	0.4	0.2
10nmFe	0.6	0.6	0.6	1.0	0.4	0.3	0.0
2nmNi	0.4	0.4	0.4	0.4	1.0	0.7	0.6
Ni0.6nm	0.4	0.4	0.4	0.3	0.7	1.0	0.6
10nmNi	0.2	0.2	0.2	0.0	0.6	0.6	1.0

Table 7.5 Correlation matrix describing the relationship between estimated performance of different catalysts, as estimated by an expert.

Constructing the covariance matrix involves incorporating the structure of the problem. This may be relatively easy, as with the covariance between discretized choices of a continuous surface.

There is a more compact way of updating our estimate of $\bar{\mu}^n$ in the presence of correlated beliefs. Let $\lambda^W = \sigma_W^2 = 1/\beta^W$ (this is basically a trick to get rid of that nasty square). Let $\Sigma^{n+1}(x)$ be the updated covariance matrix given that we have chosen to evaluate alternative

x, and let $\tilde{\Sigma}^n(x)$ be the change in the covariance matrix due to evaluating x, which is given by

$$\begin{aligned}\tilde{\Sigma}^n(x) &= \Sigma^n - \Sigma^{n+1}, \\ &= \frac{\Sigma^n e_x (e_x)^T \Sigma^n}{\Sigma^n_{xx} + \lambda^W},\end{aligned}$$

where e_x is a vector of 0s with a 1 in the position corresponding to alternative x. Now define the vector $\tilde{\sigma}^n(x)$, which gives the square root of the change in the variance due to measuring x, which is given by

$$\tilde{\sigma}^n(x) = \frac{\Sigma^n e_x}{\sqrt{\Sigma^n_{xx} + \lambda^W}}. \tag{7.77}$$

Let $\tilde{\sigma}_i(\Sigma, x)$ be the component $(e_i)^T \tilde{\sigma}(x)$ of the vector $\tilde{\sigma}(x)$, and let $\mathit{Var}^n(\cdot)$ be the variance given what we know after n experiments. We note that if we evaluate alternative x^n, then

$$\begin{aligned}\mathit{Var}^n\left[W^{n+1} - \bar{\mu}^n_{x^n}\right] &= \mathit{Var}^n\left[\mu_{x^n} + \varepsilon^{n+1}\right] \\ &= \Sigma^n_{x^n x^n} + \lambda^W. \end{aligned} \tag{7.78}$$

Next define the random variable

$$Z^{n+1} = (W^{n+1} - \bar{\mu}^n_{x^n}) / \sqrt{\mathit{Var}^n\left[W^{n+1} - \bar{\mu}^n_{x^n}\right]}.$$

We can now rewrite our expression which we first saw in chapter 3, equation (7.26) for updating our beliefs about the mean as

$$\bar{\mu}^{n+1} = \bar{\mu}^n + \tilde{\sigma}(x^n) Z^{n+1}. \tag{7.79}$$

Note that $\bar{\mu}^{n+1}$ and $\bar{\mu}^n$ are vectors giving beliefs for all alternatives, not just the alternative x^n that we tested. The knowledge gradient policy for correlated beliefs is computed using

$$\begin{aligned}X^{KG}(s) &= \arg\max_x \mathbb{E}\left[\max_i \mu_i^{n+1} \mid S^n = s\right] \\ &= \arg\max_x \mathbb{E}\left[\max_i \left(\bar{\mu}_i^n + \tilde{\sigma}_i(x^n) Z^{n+1}\right) \mid S^n, x\right].\end{aligned} \tag{7.80}$$

where Z is a scalar, standard normal random variable. The problem with this expression is that the expectation is harder to compute, but a simple algorithm can be used to compute the expectation exactly. We start by defining

$$h(\bar{\mu}^n, \tilde{\sigma}(x)) = \mathbb{E}\left[\max_i \left(\bar{\mu}_i^n + \tilde{\sigma}_i(x^n) Z^{n+1}\right) \mid S^n, x = x^n\right]. \tag{7.81}$$

Substituting (7.81) into (7.80) gives us

$$X^{KG}(s) = \arg\max_x h(\bar{\mu}^n, \tilde{\sigma}(x)). \tag{7.82}$$

Let $a_i = \bar{\mu}_i^n$, $b_i = \tilde{\sigma}_i(\Sigma^n, x^n)$, and let Z be our standard normal deviate. Now define the function $h(a, b)$ as

$$h(a, b) = \mathbb{E} \max_i \left(a_i + b_i Z\right). \tag{7.83}$$

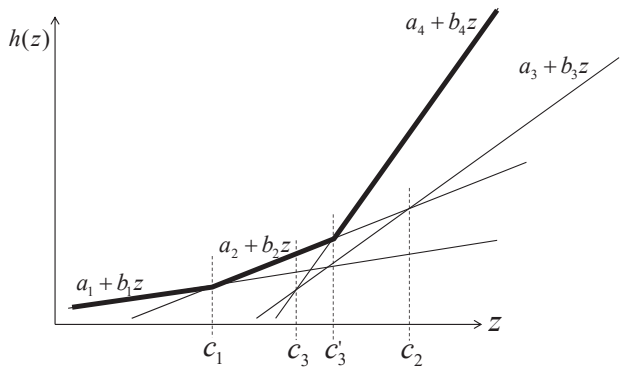


Figure 7.10 Regions of z over which different choices dominate. Choice 3 is always dominated.

Both a and b are M-dimensional vectors. Sort the elements b_i so that $b_1 \leq b_2 \leq \ldots$ so that we get a sequence of lines with increasing slopes, as depicted in figure 7.10. There are ranges for z over a particular line may dominate the other lines, and some lines may be dominated all the time (such as alternative 3).

We need to identify and eliminate the dominated alternatives. To do this we start by finding the points where the lines intersect. The lines $a_i + b_i z$ and $a_{i+1} + b_{i+1} z$ intersect at

$$z = c_i = \frac{a_i - a_{i+1}}{b_{i+1} - b_i}.$$

For the moment, we are going to assume that $b_{i+1} > b_i$. If $c_{i-1} < c_i < c_{i+1}$, then we can find a range for z over which a particular choice dominates, as depicted in figure 7.10. A line is dominated when $c_{i+1} < c_i$, at which point they are dropped from the set. Once the sequence c_i has been found, we can compute (7.80) using

$$h(a, b) = \sum_{i=1}^{M} (b_{i+1} - b_i) f(-|c_i|),$$

where as before, $f(z) = z\Phi(z) + \phi(z)$. Of course, the summation has to be adjusted to skip any choices i that were found to be dominated.

It is important to recognize that there is more to incorporating correlated beliefs than simply using the covariances when we update our beliefs after an experiment. With this procedure, we anticipate the updating before we even perform an experiment.

The ability to handle correlated beliefs in the choice of what experiment to perform is an important feature that has been overlooked in other procedures. It makes it possible to make sensible choices when our experimental budget is much smaller than the number of potential choices we have to evaluate. There are, of course, computational implications. It is relatively easy to handle dozens or hundreds of alternatives, but as a result of the matrix calculations, it becomes expensive to handle problems where the number of potential choices is in the thousands. If this is the case, it is likely the problem has special structure. For example, we might be discretizing a p-dimensional parameter surface, which suggests using a parametric model for the belief model.

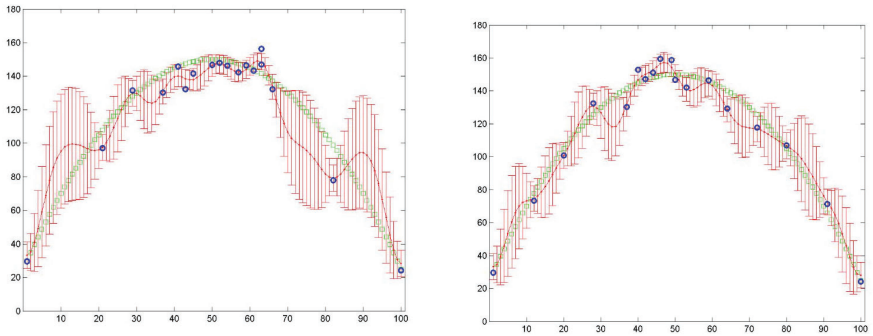


Figure 7.11 (a) Sampling pattern from knowledge gradient using independent beliefs; (b) sampling pattern from knowledge gradient using correlated beliefs.

A reasonable question to ask is: given that the correlated KG is considerably more complex than the knowledge gradient policy with independent beliefs, what is the value of using correlated KG? Figure 7.11(a) shows the sampling pattern when learning a quadratic function, starting with a uniform prior, when using the knowledge gradient with independent beliefs for the learning policy, but using correlated beliefs to update beliefs after an experiment has been run. This policy tends to produce sampling that is more clustered in the region near the optimum. Figure 7.11(b) shows the sampling pattern for the knowledge gradient policy with correlated beliefs, showing a more uniform pattern that shows a better spread of experiments.

So, the correlated KG logic seems to do a better job of exploring, but how well does it work? Figure 7.12 shows the opportunity cost for each policy, where smaller is better. For this example, the correlated KG works quite a bit better, probably due to the tendency of the correlated KG policy to do explore more efficiently.

While these experiments suggest strong support for the correlated KG policy when we have correlated beliefs, we need to also note that tunable CFA-based policies such as interval estimation or the UCB policies can also be tuned in the context of problems with correlated beliefs. The tradeoff is that the correlated KG policy does not require tuning, but is more difficult to implement. A tuned CFA policy requires tuning (which can be a challenge) but is otherwise trivial to implement. This is the classic tradeoff between a CFA policy (in the policy search class) and a DLA policy (in the lookahead class).

7.9 LEARNING IN BATCHES

There are many settings where it is possible to do simultaneous observations, effectively learning in batch. Some examples are:

- If learning is being done via computer simulation, different runs can be run in parallel.
- An automotive manufacturer looking to tune its robots can try out different ideas at different plants.
- A company can perform local test marketing in different cities, or by using ads targeted for different types of people shopping online.

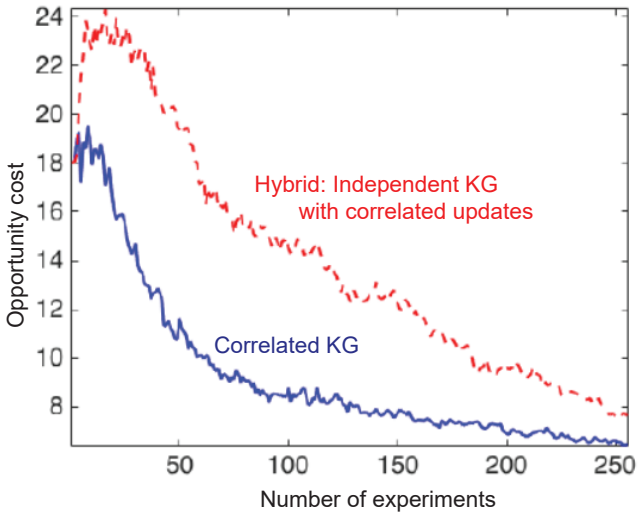


Figure 7.12 Comparison of correlated KG policy against a KG policy with independent beliefs, but using correlated updates, showing the improvement when using the correlated KG policy.

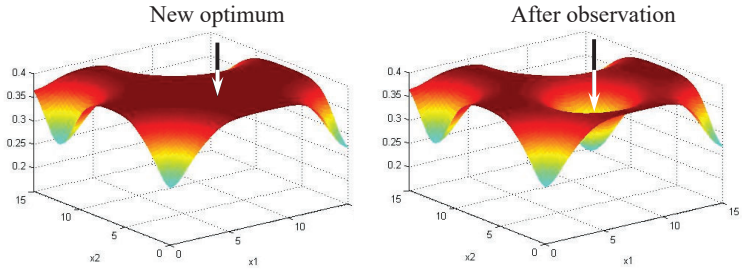


Figure 7.13 The knowledge gradient with correlated beliefs before running an experiment (left) and after (right), showing the drop in the knowledge gradient both at the point of the test, and the neighboring region, after running the experiment.

- A materials scientist looking for new materials can divide a plate into 25 squares and perform 25 experiments in batch.

When parallel testing is possible, the natural question is then: how do we determine the set of tests before we know the outcomes of the other tests? We cannot simply apply a policy repeatedly, as it might just end up choosing the same point (unless there is some form of forced randomization).

A simple strategy is to simulate a sequential learning process. That is, use some policy to determine the first test, and then either use the expected outcome, or a simulated outcome, to update the beliefs from running the test, and then repeat the process. The key is to update the beliefs after each simulated outcome. If you can perform K experiments in parallel, repeat this process K times.

Figure 7.13 shows the effect of running an experiment when using correlated beliefs when using the knowledge gradient, although the principle applies to a number of learning policies. On the left is the knowledge gradient before running the indicated experiment, and the right shows the knowledge gradient after running the experiment. As a result of using correlated beliefs, the knowledge gradient drops in the region around the first experiment, discouraging the choice of another experiment nearby. Note that what is important is where you are planning on doing the first experiment, not the outcome. This is particularly true of the knowledge gradient.

7.10 SIMULATION OPTIMIZATION*

A subcommunity within the larger stochastic search community goes by the name *simulation optimization*. This community also works on problems that can be described in the form of $\max_x \mathbb{E}F(x, W)$, but the context typically arises when x represents the design of a physical system, which is then evaluated (noisily) using discrete-event simulation. The number of potential designs $\mathcal{X}$ is typically in the range of 5 to perhaps 100. The standard approach in simulation optimization is to use a frequentist belief model, where it is generally assumed that our experimental budget is large enough for us to run some initial testing of each of the alternatives to build an initial belief.

The field of simulation-optimization has its roots in the analysis of designs, such as the layout of a manufacturing system, where we can get better results if we run a discrete event simulation model for a longer time. We can evaluate a design x more accurately by increasing the run length n_x of the simulation, where n_x might be the number of time periods, the CPU time, or the number of discrete events (e.g. customer arrivals). We assume that we have a global budget N, and we need to find n_x for each x so that

$$\sum_{x \in \mathcal{X}} n_x = N.$$

For our purposes, there is no difference between a potential design of a physical system and a policy. Searching for the best design and searching for the best policy is, algorithmically speaking, identical as long as the set of policies is not too large.

We can tackle this problem using the strategies described above (such as the knowledge gradient) if we break up the problem into a series of short simulations (say, 1 time step or 1 unit of CPU time). Then, at each iteration we have to decide which design x to evaluate, contributing to our estimate θ_x^n for design x. The problem with this strategy is that it ignores the startup time for a simulation. It is much easier to set a run length n_x for each design x, and then run the entire simulation to obtain an estimate of θ_x.

The simulation-optimization problem is traditionally formulated in a frequentist framework, reflecting the lack of prior information about the alternatives. A standard strategy is to run the experiments in two stages. In the first stage, a sample n^0 is collected for each design. The information from this first stage is used to develop an estimate of the value of each design. We might learn, for example, that certain designs seem to lack any promise at all, while other designs may seem more interesting. Rather than spreading our budget across all the designs, we can use this information to focus our computing budget across the designs that offer the greatest potential.

Step 0. Initialization:

Step 0a. Select the probability of correct selection $1-\alpha$, indifference zone parameter δ and initial sample size $n_0 \geq 2$.

Step 0b. Compute

$$\eta = \frac{1}{2}\left[\left(\frac{2\alpha}{k-1}\right)^{-2/(n_0-1)} - 1\right].$$

Step 0c. Set $h^2 = 2\eta(n_0 - 1)$.

Step 0d. Set $\mathcal{X}^0 = \mathcal{X}$ as the set of systems in contention.

Step 0e. Obtain samples W_x^m, $m = 1, \ldots, n_0$ of each $x \in \mathcal{X}^0$ and let θ_x^0 be the resulting sample means for each alternative computing using

$$\theta_x^0 = \frac{1}{n_0}\sum_{m=1}^{n_0} W_x^m.$$

Compute the sample variances for each pair using

$$\hat{\sigma}_{xx'}^2 = \frac{1}{n_0 - 1}\sum_{m=1}^{n_0}\left[W_x^m - W_{x'}^m - (\theta_x^0 - \theta_{x'}^0)\right]^2.$$

Set $r = n_0$.

Step 0f. Set $n = 1$.

Step 1. Compute

$$W_{xx'}(r) = \max\left\{0, \frac{\delta}{2r}\left(\frac{h^2\hat{\sigma}_{xx'}^2}{\delta^2} - r\right)\right\}.$$

Step 2. Refine the eligible set using

$$\mathcal{X}^n = \left\{x : x \in \mathcal{X}^{n-1} \text{ and } \theta_x^n \geq \theta_{x'}^n - W_{xx'}(r), x' \neq x\right\}.$$

Step 3. If $|\mathcal{X}^n| = 1$, stop and select the element in $\mathcal{X}^n$. Otherwise, perform an additional sample W_x^{n+1} of each $x \in \mathcal{X}^n$, set $r = r + 1$ and return to step 1.

Figure 7.14 Policy search algorithm using the indifference zone criterion, due to Nelson & Kim (2001).

7.10.1 An indifference zone algorithm

There are a number of algorithms that have been suggested to search for the best design using the indifference zone criterion, which is one of the most popular in the simulation-optimization community. The algorithm in figure 7.14 summarizes a method which successively reduces a set of candidates at each iteration, focusing the evaluation effort on a smaller and smaller set of alternatives. The method (under some assumptions) using a user-specified indifference zone of δ. Of course, as δ is decreased, the computational requirements increase.

7.10.2 Optimal computing budget allocation

The value of the indifference zone strategy is that it focuses on achieving a specific level of solution quality, being constrained by a specific budget. However, it is often the case that we are trying to do the best we can within a specific computing budget. For this purpose,

a line of research has evolved under the name *optimal computing budget allocation*, or OCBA.

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

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

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

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

After using Mn_0 observations from our budget, at each iteration we increase our allowed budget by $B^n = B^{n-1} + \Delta$ until we reach $B^N = B$. After each increment, the allocation $N_x^n,\ x \in \mathcal{X}$ is recomputed using

$$\frac{N_x^{n+1}}{N_{x'}^{n+1}} = \frac{\hat{\sigma}_x^{2,n}/(\theta_{x^n}^n - \theta_{x'}^n)^2}{\hat{\sigma}_{x'}^{2,n}/(\theta_{x^n}^n - \theta_{x'}^n)^2} \quad x \neq x' \neq x^n, \tag{7.84}$$

$$N_{x^n}^{n+1} = \hat{\sigma}_{x^n}^n \sqrt{\sum_{i=1, i\neq x^n}^{M} \left(\frac{N_x^{n+1}}{\hat{\sigma}_i^n}\right)^2}. \tag{7.85}$$

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

The complete algorithm is summarized in figure 7.15.

7.11 EVALUATING POLICIES

There are a number of ways to approach evaluating policies in the context of derivative-free stochastic search. We start by presenting alternative performance metrics and close with a discussion of alternative perspectives of optimality.

7.11.1 Alternative performance metrics*

Up to now we have evaluated the performance of a policy based on the expected value of the final or cumulative reward. However, there are many ways to evaluate a policy. Below is a list of metrics that have been drawn from different communities.

Empirical performance - We might simulate a policy K times, where each repetition involves making N observations of W. We let ω^k represent a full sample realization of

Step 0. Initialization:

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

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

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

Step 0d. Initialize $n = 1$.

Step 1. Compute

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

Compute the sample variances for each pair using

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

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

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

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

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

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

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

Figure 7.15 Optimal computing budget allocation procedure.

these N observations, which we would denote by $W^1(\omega^k), \ldots, W^N(\omega^k)$. Each sequence ω^k creates a design decision $x^{\pi,N}(\omega^k)$.

It is useful to separate the random variable W that we observe while learning from the random variable we use to evaluate a design, so we are going to let W be the random variable we observe while learning, and we are going to let $\widehat{W}$ be the random variable we use for evaluating a design. Most of the time these are the same random variable with the same distribution, but it opens the door to allowing them to be different.

Once we obtain a design $x^{\pi,N}(\omega^k)$, we then have to evaluate it by taking, say, L observations of $\widehat{W}$, which we designate by $\widehat{W}^1, \ldots, \widehat{W}^\ell, \ldots, \widehat{W}^L$. Using this notation, we would approximate the performance of a design $x^{\pi,N}(\omega^k)$ using

$$\bar{F}^\pi(\omega^k) = \frac{1}{L} \sum_{\ell=1}^{L} F(x^{\pi,N}(\omega^k), \widehat{W}^\ell)$$

We then average over all ω^k using

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

Quantiles - Instead of evaluating the average performance, we may wish to evaluate a policy based on some quantile. For example, if we are maximizing performance, we might be interested in the 10^{th} percentile, since a policy that produces good average performance may work very poorly some of the time.

Let $Q_\alpha(R)$ be the α quantile of a random variable R. Let $F^\pi = F(x^{\pi,N}, W)$ be the random variable describing the performance of policy π, recognizing that we may have uncertainty about the model (captured by S^0), uncertainty in the experiments $W^1, \ldots, W^N$ that go into the final design $x^{\pi,N}$, and then uncertainty in how well we do when we implement $x^{\pi,N}$ due to $\widehat{W}$. Now, instead of taking an expectation of F^π as we did before, we let

$$V_\alpha^\pi = Q_\alpha F(x^{\pi,N}, \widehat{W}).$$

We anticipate that there are many settings where the α quantile is more interesting than an expectation. However, we have to caution that optimizing the α quantile is much harder than optimizing an expectation.

Static regret - Deterministic setting - We illustrate static regret for deterministic problems using the context of machine learning where our decision is to choose a parameter θ that fits a model $f(x|\theta)$ to observations y. Here, "x" plays the role of data rather than a decision, although later we will get to "decide" what data to collect (confused yet?).

The machine learning community likes to evaluate the performance of a machine learning algorithm (known as a "learner") which is searching for the best parameters θ to fit some model $f(x|\theta)$ to predict a response y. Imagine a dataset $x^1, \ldots, x^n, \ldots, x^N$ and let $L^n(\theta)$ be the loss function that captures how well our function $f(x^n|\theta^n)$ predicts the response y^{n+1}, where θ^n is our estimate of θ based on the first n observations. Our loss function might be

$$L^{n+1}(x^n, y^{n+1}|\theta^n) = (y^{n+1} - f(x^n|\theta^n))^2.$$

Assume now that we have an algorithm (or policy) for updating our estimate of θ that we designate $\Theta^\pi(S^n)$, where S^n captures whatever the algorithm (or policy) needs to update θ^{n-1} to θ^n. One example of a policy is to optimize over the first n data points, so we would write

$$\Theta^\pi(S^n) = \arg\min_\theta \sum_{m=0}^{n-1} L^{m+1}(x^m, y^{m+1}|\theta)$$

Alternatively, we could use one of the gradient-based algorithms presented in chapter 5. If we fix this policy, our total loss would be

$$L^\pi = \sum_{n=0}^{N-1} L^{n+1}(x^n, y^{n+1}|\Theta^\pi(S^n)).$$

Now imagine that we pick the best value of θ, which we call θ^*, based on all the data. This requires solving

$$L^{static,*} = \min_{\theta} \sum_{n=0}^{N-1} L^{n+1}(x^n, y^{n+1}|\theta).$$

We now compare the performance of our policy, L^{π}, to our static bound, $L^{static,*}$. The difference is known as the *static regret* in the machine learning community, or the *opportunity cost* in other fields. The regret (or opportunity cost) is given by

$$R^{static,\pi} = L^{\pi} - L^{static,*}. \tag{7.86}$$

Static regret - Stochastic setting - Returning to the setting where we have to decide which alternative x to try, we now illustrate static regret in a stochastic setting, where we seek to maximize rewards ("winnings") W_x^n by trying alternative x in the n^{th} trial. Let $X^{\pi}(S^n)$ be a policy that determines the alternative x^n to evaluate given what we know after n experiments (captured by our state variable S^n). Imagine that we can generate the entire sequence of winnings W_x^n for all alternatives x, and all iterations n. If we evaluate our policy on a single dataset (as we did in the machine learning setting), we would evaluate our regret (also known as *static regret*) as

$$R^{\pi,n} = \max_{x} \sum_{m=1}^{n} W_x^m - \sum_{m=1}^{n} W^m_{X^{\pi}(S^m)}. \tag{7.87}$$

Alternatively, we could write our optimal solution at time n as

$$x^n = \arg\max_{x} \sum_{m=1}^{n} W_x^m,$$

and then write the regret as

$$R^{\pi,n} = \sum_{m=1}^{n} W^m_{x^n} - \sum_{m=1}^{n} W^m_{X^{\pi}(S^m)}.$$

The regret (for a deterministic problems) $R^{\pi,n}$ is comparing the best decision at time n assuming we know all the values W_x^m, $x \in \mathcal{X}$ for $m = 1, \ldots, n$, against what our policy $X^{\pi}(S^m)$ would have chosen given just what we know at time m (please pay special attention to the indexing). This is an instance of static regret for a deterministic problem.

In practice, W_x^m is a random variable. Let $W_x^m(\omega)$ be one sample realization for a sample path $\omega \in \Omega$ (we can think of regret for a deterministic problem as the regret for a single sample path). Here, ω represents a set of all possible realizations of W over all alternatives x, and all iterations n. Think of specifying ω as pre-generating all the observations of W that we *might* experience over all experiments. However, when we make a decision $X^{\pi}(S^m)$ at time m, we are not allowed to see any of the information that might arrive at times after m.

When we introduce uncertainty, there are now two ways of evaluating regret. The first is to assume that we are going to first observe the outcomes $W_x^m(\omega)$ for all the alternatives and the entire history $m = 1, \ldots, n$, and compare this to what our policy $X^{\pi}(S^m)$ would

have done at each time m knowing only what has happened up to time m. The result is the regret for a single sample path ω

$$R^{\pi,n}(\omega) = \max_{x(\omega)} \sum_{m=1}^{n} W^m_{x(\omega)}(\omega) - \sum_{m=1}^{n} W^m_{X^\pi(S^m)}(\omega). \tag{7.88}$$

As we did above, we can also write our optimal decision for the stochastic case as

$$x^n(\omega) = \arg\max_{x\in\mathcal{X}} \sum_{m=1}^{n} W^m_x(\omega).$$

We would then write our regret for sample path ω as

$$R^{\pi,n}(\omega) = \sum_{m=1}^{n} W^m_{x^n(\omega)}(\omega) - \sum_{m=1}^{n} W^m_{X^\pi(S^m)}(\omega).$$

Think of $x^n(\omega)$ as the best answer if we actually did know $W^m_x(\omega)$ for $m = 1,\ldots,n$, which in practice would never be true.

If we use our machine learning setting, the sample ω would be a single dataset used to fit our model. In machine learning, we typically have a single dataset, which is like working with a single ω. This is typically what is meant by a deterministic problem (think about it). Here, we are trying to design policies that will work well across many datasets.

In the language of probability, we would say that $R^{\pi,n}$ is a random variable (since we would get a different answer each time we run the simulation), while $R^{\pi,n}(\omega)$ is a sample realization. It helps when we write the argument (ω) because it tells us what is random, but $R^{\pi,n}(\omega)$ and $x^n(\omega)$ are sample realizations, while $R^{\pi,n}$ and x^n are considered random variables (the notation does not tell you that they are random - you just have to know it). We can "average" over all the outcomes by taking an expectation, which would be written

$$\mathbb{E}R^{\pi,n} = \mathbb{E}\left\{W^n_{x^n} - \sum_{m=1}^{n} W^m_{X^\pi(S^m)}\right\}.$$

Expectations are mathematically pretty, but we can rarely actually compute them, so we run simulations and take an average. Assume we have a set of sample realizations $\omega \in \hat{\Omega} = \{\omega^1,\ldots,\omega^\ell,\ldots,\omega^L\}$. We can compute an average regret (approximating expected regret) using

$$\mathbb{E}R^{\pi,n} \approx \frac{1}{L}\sum_{\ell=1}^{L} R^{\pi,n}(\omega^\ell).$$

Classical static regret assumes that we are allowed to find a solution $x^n(\omega)$ for each sample path. There are many settings where we have to find solutions before we see any data, that works well, on average, over all sample paths. This produces a different form of regret known in the computer science community as *pseudo-regret* which compares a policy $X^\pi(S^n)$ to the solution x^* that works best *on average* over all possible sample paths. This is written

$$\bar{R}^{\pi,n} = \max_x \mathbb{E}\left\{\sum_{m=1}^{n} W^n_x\right\} - \mathbb{E}\left\{\sum_{m=1}^{n} W^n_{X^\pi(S^n)}(\omega)\right\}. \tag{7.89}$$

Again, we will typically need to approximate the expectation using a set of sample paths $\hat{\Omega}$.

Dynamic regret - A criticism of static regret is that we are comparing our policy to the best decision x^* (or best parameter θ^* in a learning problem) for an entire dataset, but made after the fact with perfect information. In online settings, it is necessary to make decisions x^n (or update our parameter θ^n) using only the information available up through iteration n.

Dynamic regret raises the bar by choosing the best value θ^n that minimizes $L^n(x^{n-1}, y^n|\theta)$, which is to say

$$\theta^{*,n} = \arg\min_{\theta} L^n(x^{n-1}, y^n|\theta), \tag{7.90}$$

$$= \arg\min_{\theta}(y^n - f(x^{n-1}|\theta))^2. \tag{7.91}$$

The dynamic loss function is then

$$L^{dynamic,*} = \sum_{n=0}^{N-1} L^{n+1}(x^n, y^{n+1}|\theta^{*,n}).$$

More generally, we could create a policy Θ^π for adaptively evolving θ (equation (7.91) is an example of one such policy). In this case we would compute θ using $\theta^n = \Theta^\pi(S^n)$, where S^n is our belief state at time n (this could be current estimates, or the entire history of data). We might then write our dynamic loss problem in terms of finding the best policy Θ^π for adaptively searching for θ as

$$L^{dynamic,*} = \min_{\Theta^\pi} \sum_{n=0}^{N-1} L^{n+1}(x^n, y^{n+1}|\Theta^\pi(S^n)).$$

We then define dynamic regret using

$$R^{dynamic,\pi} = L^\pi - L^{dynamic,*}.$$

Dynamic regret is simply a performance metric using a more aggressive benchmark. It has attracted recent attention in the machine learning community as a way of developing theoretical benchmarks for evaluating learning policies.

Opportunity cost (stochastic) - Opportunity cost is a term used in the learning community that is the same as regret, but often used to evaluate policies in a stochastic setting. Let $\mu_x = \mathbb{E}F(x, \theta)$ be the true value of design x, let

$$x^* = \arg\max_x \mu_x,$$

$$x^\pi = \arg\max_x \mu_{x^{\pi,N}}.$$

So, x^* is the best design if we knew the truth, while $x^{\pi,N}$ is the design we obtained using learning policy π after exhausting our budget of N experiments. In this setting, μ_x is treated deterministically (think of this as a known truth), but x^{π_N} is random because it depends on a noisy experimentation process. The expected regret, or opportunity cost, of policy π is given by

$$R^\pi = \mu_{x^*} - \mathbb{E}\mu_{x^{\pi,N}}. \tag{7.92}$$

Competitive analysis - A strategy that is popular in the field known as *online computation* (which has nothing to do with "online learning") likes to compare the performance of a policy to the best that could have been achieved. There are two ways to measure "best." The most common is to assume we know the future. Assume we are making decisions $x^0, x^1, \ldots, x^T$ over our horizon $0, \ldots, T$. Let ω represent a sample path $W^1(\omega), \ldots, W^N(\omega)$, and let $x^{*,t}(\omega)$ be the best decision given that we know that all random outcomes (over the entire horizon) are known (and specified by ω). Finally, let $F(x^n, W^{n+1}(\omega))$ be the performance that we observe at time $t+1$. We can then create a perfect foresight (PF) policy using

$$X^{PF,n}(\omega) = \arg\max_{x^n(\omega)} \left(c^n x^n(\omega) + \max_{x^{n+1}(\omega),\ldots,x^N(\omega)} \sum_{m=n+1}^{N} c^m x^m(\omega) \right).$$

Unlike every other policy that we consider in this volume, this policy is allowed to see into the future, producing decisions that are better than anything we could achieve without this ability. Now consider some $X^\pi(S^n)$ policy that is only allowed to see the state at time S^n. We can compare policy $X^\pi(S)$ to our perfect foresight using the *competitive ratio* given by

$$\rho^\pi = \mathbb{E} \frac{\sum_{n=0}^{N-1} F(X^{\pi,n}(\omega), W^{n+1}(\omega))}{\sum_{n=0}^{N-1} F(X^{PF,n}(\omega), W^{n+1}(\omega))}$$

where the expectation is over all sample paths ω (competitive analysis is often performed for a single sample path). Researchers like to prove bounds on the competitive ratio, although these bounds are never tight.

Indifference zone selection - A variant of the goal of choosing the best alternative $x^* = \arg\max_x \mu_x$ is to maximize the likelihood that we make a choice $x^{\pi,N}$ that is almost as good as x^*. Assume we are equally happy with any outcome within δ of the best, by which we mean

$$\mu_{x^*} - \mu_{x^{\pi,N}} \leq \delta.$$

The region $(\mu_{x^*} - \delta, \mu_{x^*})$ is referred to as the *indifference zone*. Let $V^{n,\pi}$ be the value of our solution after n experiments. We require $\mathbb{P}^\pi\{\mu_{d^*} = \bar{\mu}^* | \mu\} > 1 - \alpha$ for all μ where $\mu_{[1]} - \mu_{[2]} > \delta$, and where $\mu_{[1]}$ and $\mu_{[2]}$ represent, respectively, the best and second best choices.

We might like to maximize the likelihood that we fall within the indifference zone, which we can express using

$$P^{IZ,\pi} = \mathbb{P}^\pi(V^{\pi,n} > \mu^* - \delta).$$

As before, the probability has to be computed with the appropriate Bayesian or frequentist distribution.

7.11.2 Perspectives of optimality*

In this section we review different perspectives of optimality in sequential search procedures.

Asymptotic convergence for final reward - While in practice we need to evaluate how an algorithm does in a finite budget, there is a long tradition in the analysis of algorithms to study the asymptotic performance of algorithms when using a final-reward criterion. In particular, if x^* is the solution to our asymptotic formulation in equation (7.1), we would like to know if our policy that produces a solution $x^{\pi,N}$ after N evaluations would eventually converge to x^*. That is, we would like to know if

$$\lim_{N\rightarrow\infty} x^{\pi,N} \rightarrow x^*.$$

Researchers will often begin by proving that an algorithm is asymptotically convergent (as we did in chapter 5), and then evaluate the performance in a finite budget N empirically. Asymptotic analysis generally only makes sense when using a final-reward objective.

Finite time bounds on choosing the wrong alternative - There is a body of research that seeks to bound the number of times a policy chooses a suboptimal alternative (where alternatives are often referred to as "arms" for a multiarmed bandit problem). Let μ_x be the (unknown) expected reward for alternative x, and let $W_x^n = \mu_x + \epsilon_x^n$ be the observed random reward from trying x. Let x^* be the optimal alternative, where

$$x^* = \arg\max_x \mu_x.$$

For these problems, we would define our loss function as

$$L^n(x^n) = \begin{cases} 1 & \text{If } x^n \neq x^*, \\ 0 & \text{Otherwise.} \end{cases}$$

Imagine that we are trying to minimize the cumulative reward, which means the total number of times that we do not choose the best alternative. We can compare a policy that chooses $x^n = X^\pi(S^n)$ against a perfect policy that chooses x^* each time. The regret for this setting is then simply

$$R^{\pi,n} = \sum_{m=1}^{n} L^n(X^\pi(S^n)).$$

Not surprisingly, R^π grows monotonically in n, since good policies have to be constantly experimenting with different alternatives. An important research goal is to design bounds on $R^{\pi,n}$, which is called a finite-time bound, since it applies to $R^{\pi,n}$ for finite n.

Probability of correct selection - A different perspective is to focus on the probability that we have selected the best out of a set $\mathcal{X}$ alternatives. In this setting, it is typically the case that the number of alternatives is not too large, say 10 to 100, and certainly not 100,000. Assume that

$$x^* = \arg\max_{x\in\mathcal{X}} \mu_x$$

is the best decision (for simplicity, we are going to ignore the presence of ties). After n samples, we would make the choice

$$x^n = \arg\max_{x\in\mathcal{X}} \bar{\mu}_x^n.$$

This is true regardless of whether we are using a frequentist or Bayesian estimate.

We have made the correct selection if $x^n = x^*$, but even the best policy cannot guarantee that we will make the best selection every time. Let $\mathbb{1}_{\{\mathcal{E}\}} = 1$ if the event $\mathcal{E}$ is true, 0 otherwise. We write the probability of correct selection as

$$\begin{aligned} P^{CS,\pi} &= \text{probability we choose the best alternative} \\ &= \mathbb{E}\mathbb{1}_{\{x^n = x^*\}}, \end{aligned}$$

where the underlying probability distribution depends on our experimental policy π. The probability is computed using the appropriate distribution, depending on whether we are using Bayesian or frequentist perspectives. This may be written in the language of loss functions. We would define the loss function as

$$L^{CS,\pi} = \mathbb{1}_{\{x^n \neq x^*\}}.$$

Although we use $L^{CS,\pi}$ to be consistent with our other notation, this is more commonly represented as L_{0-1} for "0-1 loss."

Note that we write this in terms of the negative outcome so that we wish to minimize the loss, which means that we have not found the best selection. In this case, we would write the probability of correct selection as

$$P^{CS,\pi} = 1 - \mathbb{E}L^{CS,\pi}.$$

Subset selection - Ultimately our goal is to pick the best design. Imagine that we are willing to choose a subset of designs S, and we would like to ensure that $P(x^* \in S) \geq 1-\alpha$, where $1/|\mathcal{X}| < 1 - \alpha < 1$. Of course, it would be idea if $|\mathcal{S}| = 1$ or, failing this, as small as possible. Let $\bar{\mu}_x^n$ be our estimate of the value of x after n experiments, and assume that all experiments have a constant and known variance σ. We include x in the subset if

$$\bar{\mu}_x^n \geq \max_{x' \neq x} \bar{\mu}_{x'}^n - h\sigma\sqrt{\frac{2}{n}}.$$

The parameter h is the $1 - \alpha$ quantile of the random variable $\max_i Z_i^n$ where Z_i^n is given by

$$Z_i^n = \frac{(\bar{\mu}_i^n - \bar{\mu}_x^n) - (\mu_i - \mu_x)}{\sigma\sqrt{2/n}}.$$

7.12 DESIGNING POLICIES

By now we have reviewed a number of solution approaches organized by our four classes of policies:

PFAs - Policy function approximations, which are analytical functions such as a linear decision rule (that has to be tuned), or the setting of the optimal price to which we add noise (the excitation policy).

CFAs - Cost function approximations, which are probably the most popular class of policy for these problems. A good example is the family of upper confidence bounding policies, such as

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

VFAs - Policies based on value function approximation, such as using Gittins indices or backward ADP to estimate the value of information.

DLAs - Policies based on direct lookaheads such as the knowledge gradient (a one-step lookahead) or kriging.

It is easy to assume that the policy we want is the policy that performs the best. This is simply not the case. A representative from a large tech company that used active learning policies extensively stated their criteria very simply:

We will use the best policy that can be computed in under 50 milliseconds.

This hints that there is more to using a policy than just its performance. We begin our discussion with a list of characteristics of good learning policies. We then raise the issue of scaling for tunable parameters, and close with a discussion of the whole process of tuning.

7.12.1 Characteristics of a policy

Our standard approach to evaluating policies is to look for the policy that performs the best (on average) according to some performance metric. In practice, the choice of policy tends to consider the following characteristics:

Performance This is our objective function which is typically written as

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

or

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

Computational complexity CPU times matter. The tech company above required that we be able to compute a policy in 50 milliseconds, while an energy company faced a limit of 4 hours.

Robustness Is the policy reliable? Does it produce consistently reliable solutions under a wide range of data inputs? This might be important in the setting of recommending prices for hotel rooms, where the policy involves learning in the field. A hotel would not want a system that recommends unrealistic prices.

Tuning The less tuning that is required, the better.

Transparency A bank might need a system that recommends whether a loan should be approved. There are consumer protection laws that protect against bias that require a level of transparency in the reasons that a loan is turned down.

Implementation complexity How hard is it to code? How likely is it that a coding error will affect the results?

The tradeoff between simplicity and complexity is particularly important. As of this writing, CFA-based policies such as upper confidence bounding are receiving tremendous attention

in the tech sector due in large part to their simplicity, as well as their effectiveness, but always at the price of introducing tunable parameters.

The problem of tuning is almost uniformly overlooked by the theory community that focuses on theoretical performance metrices such as regret bounds. Practitioners, on the other hand, are aware of the importance of tuning, but tuning has historically been an ad hoc activity, and far too often it is simply overlooked!. Tuning is best done in a simulator, but simulators are only approximations of the real world, and they can be expensive to build. We need more research into online tuning.

Lookahead policies may have no tuning, such as the knowledge gradient in the presence of concave value of information or the deterministic direct lookahead, or some tuning such as the parameter θ^{KGLA} in section 7.7.3. Either way, a lookahead policy requires a lookahead model, which introduces its own approximations. So, there is no free lunch.

7.12.2 The effect of scaling

Consider the case of two policies. The first is interval estimation, given by

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

which exhibits a unitless tunable parameter θ^{IE}. The second policy is a type of upper confidence bounding policy known in the literature as UCB-E, given by

$$X^{UCB-E,n}(S^n|\theta^{UCB-E}) = \arg\max_x \left(\bar{\mu}_x^n + \sqrt{\frac{\theta^{UCB-E}}{N_x^n}}\right),$$

where N_x^n is the number of times that we have evaluated alternative x. We note that unlike the interval estimation policy, the tunable parameter θ^{UCB-E} has units, which means that we have to search over a much wider range than we would when optimizing θ^{IE}.

Each of these parameters were tuned on a series of benchmark learning problems using the testing system called MOLTE, with the results reported in table 7.6. We see that the optimal value of θ^{IE} ranges from around 0.01 to 1.2. By contrast, θ^{UCB-E} ranges from 0.0001 to 2500.

These results illustrate the effect of units on tunable parameters. The UCB-E policy enjoys finite time bounds on its regret, but would never produce reasonable results without

Problem	IE	UCBE
Goldstein	0.0099	2571
AUF_HNoise	0.0150	0.319
AUF_MNoise	0.0187	1.591
AUF_LNoise	0.0109	6.835
Branin	0.2694	.000366
Ackley	1.1970	1.329
HyperEllipsoid	0.8991	21.21
Pinter	0.9989	0.000164
Rastrigin	0.2086	0.001476

Table 7.6 Optimal tuned parameters for interval estimation IE, and UCB-E (from Wang et al. (2016)).

tuning. By contrast, the optimal values of θ^{IE} for interval estimation vary over a narrower range, although conventional wisdom for this parameter is that it should range between around 1 and 3. If θ^{IE} is small, then the IE policy is basically a pure exploitation policy.

Parameter tuning can be difficult in practice. Imagine, for example, an actual setting where an experiment is expensive. How would tuning be done? This issue is typically ignored in the research literature where standard practice is to focus on provable qualities. We argue that despite the presence of provable properties, the need for parameter tuning is the hallmark of a heuristic. If tuning cannot be done, the actual empirical performance of a policy may be quite poor.

Bayesian policies such as the knowledge gradient do not have tunable parameters, but do require the use of priors. Just as we do not have any real theory to characterize the behavior of algorithms that have (or have not) been tuned, we do not have any theory to describe the effect of incorrect priors.

7.12.3 Tuning

An issue that will keep coming back in the design of algorithms is tuning. We will keep repeating the mantra:

The price of simplicity is tunable parameters... and tuning is hard!

We are designing policies to solve any of a wide range of stochastic search problems, but when our policy involves tunable parameters, we are creating a stochastic search problem (tuning the parameters) to solve our stochastic search problem. The hope, of course, is that the problem of tuning the parameters of our policy is easier than the search problem that our problem is solving.

What readers need to be aware of is that the performance of their stochastic search policy can be very dependent on the tuning of the parameters of the policy. In addition, the best value of these tunable parameters can depend on anything from the characteristics of a problem to the starting point of the algorithm. This is easily the most frustrating aspect of tuning of policy parameters, since you have to know when to stop and revisit the settings of your parameters.

7.13 EXTENSIONS*

This section covers a series of extensions to our basic learning problem:

- Learning in nonstationary settings
- Strategies for designing policies
- A transient learning model
- The knowledge gradient for transient problems
- Learning with large or continuous choice sets
- Learning with exogenous state information
- State-dependent vs. state-independent problems

7.13.1 Learning in nonstationary settings

Our classic "bandit" problem involves learning the value of μ_x for $x \in \mathcal{X} = \{x_1, \ldots, x_M\}$ using observations where we choose the alternative to evaluate $x^n = X^\pi(S^n)$ from which we observe

$$W^{n+1} = \mu_{x^n} + \varepsilon^{n+1}.$$

In this setting, we are trying to learn a static set of parameters μ_x, $x \in \mathcal{X}$, using a stationary policy $X^\pi(S_t)$. An example of a stationary policy for learning is upper confidence bounding, given by

$$X^{UCB}(S^n|\theta^{UCB}) = \arg\max_x \left(\bar{\mu}_x^n + \theta^{UCB} \sqrt{\frac{\log n}{N_x^n}} \right), \tag{7.93}$$

where N_x^n is the number of times we have tried alternative x over the first n experiments.

It is natural to search for a stationary policy $X^\pi(S^n)$ (that is, a policy where the *function* does not depend on time t) by optimizing an infinite horizon, discounted objective such as

$$\max_\pi \mathbb{E} \sum_{n=0}^{\infty} \gamma^n F(X^\pi(S^n), W^{n+1}). \tag{7.94}$$

In practice, truly stationary problems are rare. Nonstationarity can arise in two ways:

Finite-horizon problems - Here we are trying to optimize performance over a finite horizon $(0, N)$ for a problem where the exogenous information W_t comes from a stationary process. The objective would be given by

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

Note that we may use a stationary policy such as upper confidence bounding to solve this problem, but it would not be optimal. The knowledge gradient policy for cumulative rewards (equation (7.65))

Learning processes - x might be an athlete who gets better as she plays, or a company might get better at making a complex component.

Exogenous nonstationarities - Field experiments might be affected by weather which is continually changing.

Adversarial response - x might be a choice of an ad to display, but the market response depends on the behavior of other players who are changing their strategies. This problem class is known as "restless bandits" in bandit community.

Availability of choices - We may wish to try different people for a job, but they may not be available on any given day. This problem is known as "intermittent bandits."

7.13.2 Strategies for designing time-dependent policies

There are two strategies for handling time-dependencies:

Time-dependent policies A time-dependent policy is simply a policy that depends on time. We already saw one instance of a nonstationary policy when we derived the knowledge gradient for cumulative rewards, which produced the policy

$$X^{OLKG,n}(S^n) = \arg\max_{x\in\mathcal{X}} \left(\bar{\mu}_x^n + (N-n)\bar{\sigma}_x^n\right). \tag{7.95}$$

Here we see that not only is the state $S^n = (\bar{\mu}^n, \bar{\sigma}^n)$ time-dependent, the policy itself is time-dependent because of the coefficient $(N-n)$. The same would be true if we used a UCB policy with coefficient $\theta^{UCB,n}$, but this means that instead of learning one parameter θ^{UCB}, we have to learn $(\theta_0^{UCB}, \theta_1^{UCB}, \ldots, \theta_N^{UCB})$.

Note that a time-dependent policy is designed in advance, before any observations have been made. This can be expressed mathematically as solving the optimization problem

$$\max_{\pi^0,\ldots,\pi^N} \mathbb{E}\sum_{n=0}^{N} F(X^{\pi^n}(S^n), W^{n+1}). \tag{7.96}$$

Adaptive policies These are policies which adapt to the data, which means the function itself is changing over time. This is easiest to understand if we assume we have a parameterized policy $X^\pi(S_t|\theta)$ (such as interval estimation - see equation (12.46)). Now imagine that the market has shifted which means we would like to increase how much exploration we are doing.

We can do this by allowing the parameter θ to vary over time, which means we would write our decision policy as $X^\pi(S^n|\theta^n)$. We need logic to adjust θ^n which we depict using $\theta^{n+1} = \Theta^{\pi^\theta}(S^n)$. The function $\Theta^{\pi^\theta}(S^n)$ can be viewed as a policy (some would call it an algorithm) to adjust θ^n. Think of this as a "policy to tune a policy."

For a given policy $X^\pi(S_t|\theta^n)$, the problem of tuning the π^θ-policy would be written as

$$\max_{\pi^\theta} \mathbb{E}\sum_{n=0}^{N} F(X^\pi(S^n|\Theta^\pi(S^n)), W^{n+1}).$$

We still have to choose the best decision policy $X^\pi(S^n|\theta^n)$. We could write the combined problem as

$$\max_{\pi^\theta}\max_{\pi} \mathbb{E}\sum_{n=0}^{N} F(X^\pi(S^n|\Theta^\pi(S^n)), W^{n+1}).$$

Both policies π^θ, which determines $\Theta^{\pi^\theta}(S^n)$, and π, which determines $X^\pi(S^n|\theta^n)$ have to be determined offline, but the decision policy is being tuned adaptively while in the field (that is, "online").

7.13.3 A transient learning model

We first introduced this model in section 3.11 where the true mean varies over time. It is most natural to talk about nonstationary problems in terms of varying over time t, but we will stay with our counter index n for consistency.

according to the model

$$\mu^{n+1} = M^n \mu^n + \varepsilon^{\mu,n+1},$$

where $\varepsilon^{\mu,n+1}$ is a random variable with distribution $N(0, \sigma^2_\mu)$, which means that $\mathbb{E}\{\mu^{n+1}|\mu^n\} = M^n\mu^n$. The matrix M^n is a diagonal matrix that captures predictable changes (e.g. where the means are increasing or decreasing predictably). If we let M^n be the identity matrix, then we have the simpler problem where the changes in the means have mean 0 which means that we expect $\mu^{n+1} = \mu^n$. However, there are problems where there can be a predictable drift, such as estimating the level of a reservoir changing due to stochastic rainfall and predictable evaporation. We then make noisy observations of μ^n using

$$W^n = M^n\mu^n + \varepsilon^n.$$

It used to be that if we did not observe an alternative x' that our belief $\bar{\mu}^t_{x'}$ did not change (and of course, nor did the truth). Now, the truth may be changing, and to the extent that there is predictable variation (that is, M^n is not the identity matrix), then even our beliefs may change.

The updating equation for the mean vector is given by

$$\bar{\mu}^{t+1}_x = \begin{cases} M^n_x \bar{\mu}^n_x + \frac{W^{n+1} - M^n_x \bar{\mu}^n_x}{\sigma^2_\varepsilon + \Sigma^n_{xx}} \Sigma^n_{xx} & \text{If } x^n = x, \\ M^n_x \bar{\mu}^n_x & \text{Otherwise.} \end{cases} \tag{7.97}$$

To describe the updating of Σ^n, let Σ^n_x be the column associated with alternative x, and let e_x be a vector of 0's with a 1 in the position corresponding to alternative x. The updating equation for Σ^n can then be written

$$\Sigma^{t+1}_x = \begin{cases} \Sigma^n_x - \frac{(\Sigma^n_x)^T \Sigma^n_x}{\sigma^2_\varepsilon + \Sigma^n_{xx}} e_x & \text{If } x^n = x, \\ \Sigma^n_x & \text{Otherwise.} \end{cases} \tag{7.98}$$

These updating equations can play two roles in the design of learning policies. First, they can be used in a lookahead policy, as we illustrate next with the knowledge gradient (a one-step lookahead policy). Alternatively, they can be used in a simulator for the purpose of doing policy search for the best PFA or CFA.

7.13.4 The knowledge gradient for transient problems

To compute the knowledge gradient, we first compute

$$\begin{aligned} \tilde{\sigma}^{2,n}_x &= \text{The conditional change in the variance of } \bar{\mu}^{n+1}_x \text{ given what we know now,} \\ &= \mathit{Var}(\bar{\mu}^{n+1}_x|\bar{\mu}^n) - \mathit{Var}(\bar{\mu}^n), \\ &= \mathit{Var}(\bar{\mu}^{n+1}_x|\bar{\mu}^n), \\ &= \tilde{\Sigma}^n_{xx}. \end{aligned}$$

We can use $\tilde{\sigma}^n_x$ to write the updating equation for $\bar{\mu}^n$ using

$$\bar{\mu}^{n+1} = M^n\bar{\mu}^n + \tilde{\sigma}^n_x Z^{n+1} e_p,$$

where $Z^{n+1} \sim N(0,1)$ is a scalar, standard normal random variable.

We now present some calculations that parallel the original knowledge gradient calculations. First, we define ζ_{tx} as we did before

$$\zeta_x^n = -\left|\frac{\bar{\mu}_x^n - \max_{x' \neq x} \bar{\mu}_{x'}^n}{\tilde{\sigma}_x^n}\right|.$$

This is defined for our stationary problem. We now define a modified version that we call ζ_x^M that is given by

$$\zeta_x^{M,n} = M^n \zeta_x^n.$$

We can now compute the knowledge gradient for nonstationary truths using a form that closely parallels the original knowledge gradient,

$$\begin{aligned} \nu_x^{KG-NS,n} &= \tilde{\sigma}_x^n \left(\zeta_x^{M,n}\Phi(\zeta_x^{M,n}) + \phi(\zeta_x^{M,n})\right) && (7.99) \\ &= \tilde{\sigma}_x^n \left(M^n\zeta_x^n\Phi(M^n\zeta_x^n) + \phi(M^n\zeta_x^n)\right). && (7.100) \end{aligned}$$

It is useful to compare this version of the knowledge gradient to the knowledge gradient for our original problem with static truths. If M^n is the identity matrix, then this means that the truths μ^n are not changing in a predictable way; they might increase or decrease, but on average μ^{n+1} is the same as μ^n. When this happens, the knowledge gradient for the transient problem is the same as the knowledge gradient when the truths are not changing at all.

So, does this mean that the problem where the truths are changing is the same as the one where they remain constant? Not at all. The difference arises in the updating equations, where the precision of alternatives x' that are not tested decrease, which will make them more attractive from the perspective of information collection.

7.13.5 Learning with large or continuous choice sets

There are many problems where our choice set $\mathcal{X}$ is either extremely large or continuous (which means the number of possible values is infinite). For example:

EXAMPLE 7.4

A website advertising movies has the space to show 10 suggestions out of hundreds of movies within a particular genre. The website has to choose from all possible combinations of 10 movies out of the population.

EXAMPLE 7.5

A scientist is trying to choose the best from a set of over 1000 different materials, but has a budget to only test 20.

EXAMPLE 7.6

A bakery chef for a food producer has to find the best proportions of flour, milk, yeast, and salt.

EXAMPLE 7.7

A basketball coach has to choose the best five starting players from a team of 12. It takes approximately half a game to draw conclusions about the performance of how well five players work together.

Each of these examples exhibit large choice sets, particularly when evaluated relative to the budget for running experiments. Such situations are surprisingly common. We can handle these situations using a combination of strategies:

Generalized learning The first step in handling large choice sets is using a belief model that provides for a high level of generalization. This can be done using correlated beliefs for lookup table models, and parametric models, where we only have to learn a relatively small number of parameters (which we hope is smaller than our learning budget).

Sampled actions Whether we have continuous actions or large (often multidimensional) actions, we can create smaller problems by just using a sampled set of actions, just as we earlier used sampled beliefs about a parameter vector θ.

Action sampling is simply another use of Monte Carlo simulation to reduce a large set to a small one, just as we have been doing when we use Monte Carlo sampling to reduce large (often infinite) sets of outcomes of random variables to smaller, discrete sets. Thus, we might start with the optimization problem

$$F^* = \max_{x \in \mathcal{X}} \mathbb{E}_W F(x, W).$$

Often the expectation cannot be computed, so we replace the typically large set of outcomes of W, represented by some set Ω, with a sampled set of outcomes $\hat{\Omega} = \{w_1, w_2, \ldots, w_K\}$, giving us

$$\bar{F}^K = \max_{x \in \mathcal{X}} \frac{1}{K} \sum_{k=1}^{K} F(x, w_k).$$

When $\mathcal{X}$ is too large, we can play the same game and replace it with a random sample $\hat{\mathcal{X}} = \{x_1, \ldots, x_L\}$, giving us the problem

$$W^{K,L} = \max_{x \in \hat{\mathcal{X}}} \frac{1}{K} \sum_{k=1}^{K} F(x, w_k). \tag{7.101}$$

Section 4.3.2 provides results that demonstrate that the approximation $\bar{F}^K$ converges quite quickly to F^* as K increases. We might expect a similar result from $W^{K,L}$ as L increases, although there are problems where it is not possible to grow L past a certain amount. For example, see equation (7.76) for our sampled belief model, which becomes computationally challenging if the number of sampled values of θ is too large.

A strategy for overcoming this limitation is to periodically drop, say, $L/2$ elements of $\mathcal{X}$ (based on the probabilities p_k^n), and then go through a process of randomly generating new values and adding them to the set until we again have L elements. We may even be able to obtain an estimate of the value of each of the new alternatives before running any new experiments. This can be done using the following:

- If we have a parametric belief model, we can estimate a value of x using our current estimate of θ. This could be a point estimate, or distribution $(p_k^n)_{k=1}^K$ over a set of possible values $\theta_1, \ldots, \theta_K$.
- If we are using lookup tables with correlated beliefs, and assuming we have access to a correlation function that gives us $Cov(F(x), F(x'))$ for any pair x and x', we can construct a belief from experiments we have run up to now. We just have to rerun the correlated belief model from chapter 3 including the new alternative, but without running any new experiments.
- We can always use nonparametric methods (such as kernel regression) to estimate the value of any x from the observations we have made so far, simply by smoothing over the new point. Nonparametric methods can be quite powerful (hierarchical aggregation is an example, even though we present it alongside lookup table models in chapter 3), but they assume no structure and as a result need more observations.

Using these estimates, we might require that any newly generated alternative x be at least as good as any of the estimates of values in the current set. This process might stop if we cannot add any new alternatives after testing some number M.

7.13.6 Learning with exogenous state information - the contextual bandit problem

The original statement of our basic stochastic optimization problem (in its asymptotic form),

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

is looking for a solution in the form of a deterministic decision x^*. We then proposed that a better form was

$$\max_x \mathbb{E}\{F(x, W)|S^0\}. \tag{7.102}$$

Again, we assume that we are looking for a single decision x^*, although now we have to recognize that technically, this decision is a function of the initial state S^0.

Now consider an adaptive learning process where a new initial state S^0 is revealed each time we try to evaluate $F(x, W)$. This changes the learning process, since each time we observe $F(x, W)$ for some x and a sampled W, what we learn has to reflect that it is in the context of the initial state S^0. Some illustrations of this setting are:

■ **EXAMPLE 7.8**

Consider a newsvendor problem where S^0 is the weather forecast for tomorrow. We know that if it is raining or very cold, that sales will be lower. We need to find an optimal order decision that reflects the weather forecast. Given the forecast, we make a decision of how many newspapers to stock, and then observe the sales.

EXAMPLE 7.9

A patient arrives to a hospital with a complaint, and a doctor has to make treatment decisions. The attributes of the patient represent initial information that the patient provides in the form of a medical history, then a decision is made, followed by a random outcome (the success of the treatment).

In both of these examples, we have to make our decision given advance information (the weather, or the attributes of the patient). Instead of finding a single optimal solution x^*, we need to find a function $x^*(S^0)$. This function is a form of policy (since it is a mapping of state to action).

This problem was first studied as a type of multiarmed bandit problems, which we first introduced in chapter 2. In this community, these are known as *contextual bandit problems*, but as we show here, when properly modeled this problem is simply an instance of a *state dependent* sequential decision problem.

We propose the following model of contextual problems. First, we let B_t be our belief state at time t that captures our belief about the function $F(x) = \mathbb{E}F(x, W)$ (keep in mind that this is distributional information). We then model two types of exogenous information:

Exogenous information - W_t^e This is information that arrives before we make a decision (this would be the weather in our newsvendor problem, or the attributes of the patient before making the medical decision).

Outcome W_t^o This is the information that arrives as a result of a decision, such as how the patient responds to a drug.

Using this notation, the sequencing of information, belief states and decisions is

$$(B^0, W^{e,0}, x^0, W^{o,1}, B^1, W^{e,1}, x^1, W^{o,2}, B^2, \ldots).$$

We have written the sequence $(W^{o,n}, B^n, W^{e,n})$ to reflect the logical progression where we first learn the outcome of a decision $W^{o,n}$, then update our belief state producing B^n, and then observe the new exogenous information $W^{e,n}$ before making decision x^n. However, we can write $W^n = (W^{o,n}, W^{e,n})$ as the exogenous information, which leads to a new state $S_t = (B^n, W^{e,n})$.

This change of variables, along with defining $S^0 = (B^0, W^{e,0})$, gives us our usual sequence of states, actions and new information that we can write as

$$(S^0 = (B^0, W^{e,0}), x^0, W^{o,1}, B^1 = B^M(B^0, x^0, W^1 = (W^{e,1}, W^{o,1})), S^1 = (B^1, W^{e,1}), x^1, W^{o,2}, B^2 = B^M(B^1, x^1, W^2 = (W^{e,2}, W^{o,2})), \ldots).$$

This, then, is the same as our basic sequence

$$(S^0, x^0, W^1, S^1, x^1, S^2, \ldots, S^n, x^n, W^{n+1}, \ldots).$$

Our policy $X^{\pi,n}(S^n)$ will now depend on both our belief state B^n about $\mathbb{E}F(x, W)$, as well as the new exogenous information $W^{e,n}$.

So why is this an issue? Simply put, pure learning problems are easier than state-dependent problems. In particular, consider one of the popular CFA policies such as upper confidence bounding or interval estimation. Instead of learning $\bar{\mu}_x^n$, we have to learn $\bar{\mu}_x^n(W^e)$. For example, if $\bar{\mu}_x^n$ describes the reduction in blood sugar from using drug x, we now have to learn the reduction in blood sugar for drug x for a patient with attributes $W^{e,n}$.

In other words, exogenous state information makes the learning more complex. If we are solving a problem where the exogenous information is weather, we might be able to describe weather using a handful of states (cold/hot, dry/rainy). However, if the exogenous information is the attributes of a patient, then it could have many dimensions. This is problematic if we are using a lookup table representation (as we might with weather), but perhaps we are just using a parametric model.

As an illustration, assume that we are deciding on the bid for an ad. The probability that a customer clicks on the ad depends on our bid b, and is given by the logistics curve:

$$p(b|\theta) = \frac{e^{U(b|\theta)}}{1+e^{U(b|\theta)}}, \tag{7.103}$$

where $U(b|\theta)$ is a linear model given by

$$U(b|\theta) = \theta_0 + \theta_1 b.$$

Now assume we are given additional information that arrives in W_t^e that provides attributes of the consumer as well as attributes of the ad. Let a_t capture this vector of attributes (this means that $W_t^e = a_t$). Then this has the effect of changing our utility function to

$$U(b|a,\theta) = \theta_0 + \theta_1 b + \theta_2 a_1 + \theta_3 a_2 + \ldots.$$

As we can see, if we are using a parametric model, the additional attributes expands the number of features in $U(b|a,\theta)$, which would increase the number of observations required to estimate the vector of coefficients θ. The number of observations needed depends on the number of parameters, and the level of noise in the data.

7.13.7 State-dependent vs. state-independent problems

We are going to spend the rest of this book on what we call "state-dependent problems," which refers to settings where the *problem* depends on the state variable. To illustrate, consider a simple newsvendor problem

$$\max_x F(x) = \mathbb{E}_W\big(p\min\{x,W\} - cx\big). \tag{7.104}$$

Assume we do not know the distribution of W, but we can collect information by choosing x^n, then observing

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

We can then use the observation $\hat{F}^{n+1}$ to produce an updated estimate $\bar{F}^{n+1}(x)$. The parameters describing the approximation $\bar{F}^n(x)$ make up our belief state B^n, which for this problem represents the only state variables. The goal is to explore different values of x to develop a good approximation $\bar{F}^n(x)$ to help choose the best value of x.

Now assume that the prices change each period, and that we are given the price p^n just before we make our choice x^n. The price p^n is a form of exogenous information, which means that instead of trying to find the best x, we are trying to find the best function $x(p)$. Now we have to decide what type of function we want to use to represent $x(p)$ (lookup table? a parametric function of p?).

Finally, assume that we have to choose product from inventory to satisfy the demand W, where R^n is our inventory. Assume that we have to observe $x^n \leq R^n$, and that the inventory is updated according to

$$R^{n+1} = R^n - \min\{x^n, W^{n+1}\} + \max\{0, x^n - W^{n+1}\}.$$

Now our decision x^n at time n affects our state R^{n+1}. For this problem, our state variable is given by

$$S^n = (R^n, p^n, B^n).$$

A special case of of a state-dependent problem was the learning problem we saw in section 7.13.6, since the problem depends on the exogenous information $W^{e,n}$. This is a type of state-dependent problem, but decisions only affect the belief; the exogenous information $W^{e,n+1}$ is not affected by the decisions x^n. This quality means that this is closer to a learning problem than the broader class of state-dependent problems.

State-dependent problems may or may not involve a belief state, but will involve information other than a belief (which is what makes them state-dependent problems). A major problem class includes problems that involve the management of resources. A simple example involves managing a vehicle moving over a graph, where the decision changes the location of the vehicle.

We will show in the remainder of the book that we can approach these more complex problems with the same five-element modeling framework that we first introduced in chapter 2, and again in this chapter. Also, we will design policies using the same four classes of policies that were covered here. What changes is the choice of which policies work best.

7.14 BIBLIOGRAPHIC NOTES

Section 7.1 - The earliest paper on derivative-free stochastic search is the seminal paper (Box & Wilson 1951), which interestingly appeared in the same year as the original paper for derivative-based stochastic search (Robbins & Monro 1951).

Section 7.2.1 - Our formulation of derivative-free stochastic search was first suggested in Powell (2019). Of particular value is writing out the objective function for evauating policies in an explicit way; perhaps surprisingly, this is often (although not always) overlooked. We are not aware of another reference formulating stochastic search problems as formal optimization problems searching for optimal policies.

Section 7.1.4 - This is the first time in writing that the equivalence of these four classes of problems have been observed.

Section 7.3 - The idea of using all four classes of policies for pure learning problems was first suggested in Powell (2019), but this book is the first to illustrate this idea in a comprehensive way.

Section 7.5 - There is by now an extensive literature in the reinforcement learning community using what are generally referred to as "upper confidence bounding" policies, which we classify under the heading of parametric cost function approximations. A nice introduction to these learning strategies is contained in Kaelbling (1993) and Sutton & Barto (2018). Thrun (1992) contains a good discussion of exploration in the learning process, which is achieved by the "uncertainty bonus" in UCB policies. The discussion of Boltzmann exploration and epsilon-greedy exploration is based on Singh et al. (2000). The upper confidence bound is due to Lai & Robbins (1985). We use the version of the UCB rule given in Lai (1987). The UCB1 policy is given in Auer et al. (2002). Analysis of UCB policies are given in Lai & Robbins (1985), as well as Chang et al. (2007).

For a nice review of Bayesian optimization see Frazier (2018).

Interval estimation is due to Kaelbling (1993) (interval estimation today is viewed (correctly) as just another form of upper confidence bounding).

See Russo et al. (2017) for a nice tutorial on Thompson sampling, which was first introduced in 1933 (Thompson (1933)).

Section 7.6 - DeGroot (1970) was the first to express pure learning problems (known at the time as multiarmed bandit problems) using Bellman's optimality equation, although it was computationally intractable. Gittins & Jones (1974) was the first to propose a decomposition of discounted infinite horizon learning problems into dynamic programs for each arm (hence of much lower dimensionality). This result produced an explosion of research into what became known as "Gittins indices" (or simply "index policies"). See Gittins (1979), Gittins (1981), and Gittins (1989). Whittle (1983) and Ross (1983) provide very clear tutorials on Gittins indices, helping to launch an extensive literature on the topic (see, for example, Lai & Robbins (1985), Berry & Fristedt (1985), and Weber (1992)). The work on approximating Gittins indices is due to Brezzi & Lai (2002), Yao (2006) and Chick & Gans (2009). In 2011 Gittins' former student, Kevin Glazebrook, came out with a "second edition" of Gittins' original book (Gittins et al. (2011)). The book was actually entirely new.

Index policies are limited to discounted, infinite horizon problems since the "index," which is related to the Lagrange multiplier on the coupling constraint requiring that we try at most one arm, needs to be independent of time. It is possible, however, to use the tools of approximate dynamic programming (in particular backward dynamic programming, described in chapter 15) to approximate the value functions around the belief state. This idea was developed by a former student (Weidong Han), but never published.

Section 7.7.2 - There are a variety of strategies based on the idea of approximating the value of one or more experiments. There is by now an extensive line of research based on the principle of the knowledge gradient, which we review in section 7.8 (see the bibliographic notes below). Sequential kriging optimization was proposed by Huang et al. (2006). Stein (1999) provides a thorough introduction to the field of kriging, which evolved from the field of spatial statistics.

An example of a restricted lookahead policy is the KG(*) policy proposed in Frazier & Powell (2010) to overcome the potential nonconcavity in the value of information.

The deterministic multiperiod lookahead was work performed jointly with graduate student Ahmet Duzgun, but was never published. It is presented here just to illustrate the range of different policies that can be tried.

The idea of using a decision tree to evaluate the value of information is standard material in the decision sciences (see, for example, Skinner (1999).

Section 7.7.5 - The hitting example in section was taken from Powell & Ryzhov (2012).

Section 7.8 - The knowledge gradient policy for normally distributed rewards and independent beliefs was introduced by Gupta & Miescke (1996), and subsequently analyzed in greater depth by Frazier et al. (2008). The knowledge gradient for correlated beliefs was introduced by Frazier et al. (2009). The adaptation of the knowledge gradient for online problems is due to Ryzhov & Powell (2009). A fairly

thorough introduction to the knowledge gradient policy is given in Powell & Ryzhov (2012) (as of this writing, a partially finished second edition is available for download from `http://castlelab.princetonedu/orf-418/`. Portions of this section are adapted from material in Powell & Ryzhov (2012).

Section 7.10 - There is an advanced field of research within the simulation community that has addressed the problem of using simulation (in particular, discrete event simulation) to find the best setting of a set of parameters that controls the behavior of the simulation. An early survey is given by Bechhofer et al. (1995); a more recent survey can be found in Fu et al. (2007). Kim et al. (2005) provides a nice tutorial overview of methods based on ordinal optimization. Other important contributions in this line include Hong & Nelson (2006) and Hong & Nelson (2007). Most of this literature considers problems where the number of potential alternatives is not too large. Nelson et al. (2001) considers the case when the number of designs is large. Ankenman et al. (2009) discusses the use of a technique called kriging, which is useful when the parameter vector x is continuous. The literature on optimal computing budget allocation is based on a series of articles originating with Chen (1995), and including Chen et al. (1997, 1998), and Chen et al. (2000). Chick et al. (2001) introduces the $LL(B)$ strategy which maximizes the linear loss with measurement budget B. He et al. (2007) introduce an OCBA procedure for optimizing the expected value of a chosen design, using the Bonferroni inequality to approximate the objective function for a single stage. A common strategy in simulation is to test different parameters using the same set of random numbers to reduce the variance of the comparisons. Fu et al. (2007) apply the OCBA concept to measurements using common random numbers. The field of simulation-optimization continues to evolve. For a more modern overview of the scope of activities, see Fu (2014).

Section 7.11.1 - The list of different objective functions is taken from Powell & Ryzhov (2012)[Chapter 6].

EXERCISES

Review questions

7.1 Explain in words each of the three nested expectations in equation (7.2).

7.2 Why do we go from maximizing over x in our original stochastic search problem in equation (7.1) to maximizing over policies π in equation (7.2)?

7.3 What is the meaning of "bandit" and "arms" in multi-armed bandit problems?

7.4 What is meant by passive learning and active learning? Why is derivative-free stochastic search an active learning problem?

7.5 State in words the information that would be needed in the state variable when describing a search algorithm for derivative-free stochastic search.

7.6 Which of the four classes of policies are used in the derivative-based stochastic search algorithms that we described in chapter 5? Which of the four classes of policies are described in this chapter for derivative-free stochastic search? Can you explain why there is the difference between derivative-based and derivative-free settings?

7.7 Give an example of a PFA-based policy for derivative-free stochastic search.

7.8 Give an example of a CFA-based policy for stochastic search.

7.9 State mathematically the definition of the knowledge gradient, and state in words what it is doing.

7.10 The knowledge gradient policy is a one-step lookahead that finds the value of one more experiment. Under what conditions does this approach fail?

7.11 What is meant by a restricted multi-step lookahead?

7.12 Give both the final-reward and cumulative reward objectives for learning problems.

7.13 Define the objective function that minimizes expected static regret.

7.14 What is meant by the indifference zone?

Modeling questions

7.15 Consider the problem of finding the best in a set of discrete choices $\mathcal{X} = \{x_1, \ldots, x_M\}$. Assume that for each alternative you maintain a lookup table belief model, where $\bar{\mu}_x^n$ is your estimate of the true mean μ_x, with precision β_x^n. Assume that your belief about μ_x is Gaussian, and let $X^\pi(S^n)$ be a policy that specifies the experiment $x^n = X^\pi(S^n)$ that you will run next, where you will learn $W_{x^n}^{n+1}$ which you will use to update your beliefs.

a) Formulate this learning problem as a stochastic optimization problem. Define your state variable, decision variable, exogenous information, transition function and objective function.

b) Specify three possible policies, with no two from the same policy class (PFA, CFA, VFA and DLA).

7.16 Section 7.3 introduces four classes of policies for derivative-free stochastic search, a concept that was not discussed when we introduced derivative-based stochastic search in chapter 5. In which of the four classes of policies would you classify a stochastic gradient algorithm? Explain, and describe a key step in the design of stochastic gradient algorithms that is explained by your choice of policy class.

7.17 A newsvendor problem where the demand distribution W is known is a static problem. When we use learning, it is a fully sequential problem. Assume we are using a derivative-based stochastic gradient algorithm from chapter 5 with a deterministic, harmonic stepsize rule. Model this system as a fully sequential problem assuming you are limited to N iterations.

7.18 Assume we are using a quadratic approximation to approximate the expected profit of a newsvendor problem:

$$F(x_t) = \mathbb{E}\left\{p \min\{x_t, W_{t+1}\} - cx_t\right\}$$

Assume you are going to be using recursive least squares to update your quadratic belief model

$$\bar{F}_t(x|\bar{\theta}_t) = \bar{\theta}_{t0} + \bar{\theta}_{t1}x + \bar{\theta}_{t2}x_t^2.$$

Further assume that you are going to choose your decision using an excitation policy of the form

$$X^{\pi}(S_t|\bar{\theta}_t) = \arg\max_{x_t} \bar{F}_t(x|\bar{\theta}_t) + \varepsilon_{t+1},$$

where $\varepsilon_{t+1} \sum N(0, \sigma_{\varepsilon}^2)$. Model this learning problem as a sequential decision problem. What class of policy are you using? What are the tunable parameters?

Computational exercises

7.19 Table 7.12 shows the priors $\bar{\mu}^n$ and the standard deviations σ^n for five alternatives.

a) Three of the alternatives have the same standard deviation, but with increasing priors. Three have the same prior, but with increasing standard deviations. Using only this information, state any relationships that you can between the knowledge gradients for each alternative. Note that you will not be able to completely rank all the alternatives.

b) Compute the knowledge gradient for each alternative assuming that $\sigma^W = 4$.

Choice	$\bar{\mu}^n$	σ^n
1	3.0	8.0
2	4.0	8.0
3	5.0	8.0
4	5.0	9.0
5	5.0	10.0

Table 7.7 Priors for exercise 9.0

7.20 You have to find the best of five alternatives. After n experiments, you have the data given in the table below. Assume that the precision of the experiment is $\beta^W = 0.6$.

Choice	θ^n	β^n	β^{n+1}	$\tilde{\sigma}$	$\max_{x' \neq x} \theta^n_{x'}$	ζ	$f(\zeta)$	ν_x^{KG}
1	3.0	0.444	1.044	1.248	6	-2.404	0.003	0.003
2	5.0	0.160	0.760	2.321	6	-0.431	0.220	0.511
3	6.0	0.207	0.807	2.003	5	-0.499	0.198	0.397
4	4.0	0.077	?	?	?	?	?	?
5	2.0	0.052	0.652	4.291	6	-0.932	0.095	0.406

a) Give the definition of the knowledge gradient, first in plain English and second using mathematics.

b) Fill in the missing entries for alternative 4 in table 7.20. Be sure to clearly write out each expression and then perform the calculation. For the knowledge gradient ν_x^{KG}, you will need to use a spreadsheet (or Matlab) to compute the normal distribution.

c) Now assume that we have an online learning problem. We have a budget of 20 experiments, and the data in the table above shows what we have learned after three experiments. Assuming no discounting, what is the online knowledge gradient for alternative 2? Give both the formula and the number.

7.21 You have to find the best of five alternatives. After n experiments, you have the data given in the table 7.21 below. Assume that the precision of the experiment is $\beta^W = 0.6$.

Alternative	$\bar{\mu}^n$	$\bar{\sigma}^n$	$\tilde{\sigma}$	ζ	$f(\zeta)$	KG index
1	4.0	2.5	2.321	-0.215	0.300	0.696
2	4.5	3.0	?	?	?	?
3	4.0	3.5	3.365	-0.149	0.329	1.107
4	4.2	4.0	3.881	-0.077	0.361	1.401
5	3.7	3.0	2.846	-0.281	0.274	0.780

a) Give the definition of the knowledge gradient, first in plain English and second using mathematics.

b) Fill in the missing entries for alternative 2 in table 7.21. Be sure to clearly write out each expression and then perform the calculation. For the knowledge gradient ν_x^{KG}, you will need to use a spreadsheet (or Matlab) to compute the normal distribution.

c) Now assume that we have an online learning problem. We have a budget of 20 experiments, and the data in the table above shows what we have learned after three experiments. Assuming no discounting, what is the online knowledge gradient for alternative 2? Give both the formula and the number.

7.22 You have three alternatives, with priors (mean and precision) as given in the first line of the table below. You then observe each of the alternatives in three successive experiments, with outcomes shown in the table. All observations are made with precision $\beta^W = 0.2$. Assume that beliefs are independent.

Iteration	A	B	C
Prior (μ_x^0, β_x^0)	(32,0.2)	(24,0.2)	(27,0.2)
1	36	-	-
2	-	-	23
3	-	22	-

Table 7.8 Three observations, for three alternatives, given a normally distributed belief, and assuming normally distributed observations.

a) Give the objective function (algebraically) for offline learning (maximizing final reward) if you have a budget of three experiments, and where you evaluate the policy using the truth (as you would do in a simulator).

b) Give the numerical value of the policy that was used to generate the choices that created table 7.8, using our ability to use the simulated truth (as you have done in your homeworks). This requires minimal calculations (which can be done without a calculator).

c) Now assume that you need to run experiments in an online (cumulative reward) setting. Give the objective function (algebraically) to find the optimal policy for online learning (maximizing cumulative reward) if you have three experiments. Using the numbers in the table, give the performance of the policy that generated the choices that were made. (This again requires minimal calculations.)

7.23 There are four paths you can take to get to your new job. On the map, they all seem reasonable, and as far as you can tell, they all take 20 minutes, but the actual times vary quite a bit. The value of taking a path is your current estimate of the travel time on that path. In the table below, we show the travel time on each path if you had travelled that path. Start with an initial estimate of each value function of 20 minutes with your tie-breaking rule to use the lowest numbered path. At each iteration, take the path with the best estimated value, and update your estimate of the value of the path based on your experience. After 10 iterations, compare your estimates of each path to the estimate you obtain by averaging the "observations" for each path over all 10 days. Use a constant stepsize of 0.20. How well did you do?

	Paths			
Day	1	2	3	4
1	37	29	17	23
2	32	32	23	17
3	35	26	28	17
4	30	35	19	32
5	28	25	21	26
6	24	19	25	31
7	26	37	33	30
8	28	22	28	27
9	24	28	31	30
10	33	29	17	29

7.24 Assume you are considering five options. The actual value μ_d, the initial estimate $\bar{\mu}_d^0$ and the initial standard deviation $\bar{\sigma}_d^0$ of each $\bar{\mu}_d^0$ are given in table 7.9. Perform 20 iterations of each of the following algorithms:

(a) Interval estimation using $\theta^{IE} = 2$.

(b) The upper confidence bound algorithm using $\theta^{UCB} = 6$.

(c) The knowledge gradient algorithm.

(d) Pure exploitation.

(e) Pure exploration.

Each time you sample a decision, randomly generate an observation $W_d = \mu_d + \sigma^\varepsilon Z$ where $\sigma^\varepsilon = 1$ and Z is normally distributed with mean 0 and variance 1. [Hint: You can generate random observations of Z in Excel by using *=NORMSINV(RAND())*.]

Decision	μ	$\bar{\theta}^0$	$\bar{\sigma}^0$
1	1.4	1.0	2.5
2	1.2	1.2	2.5
3	1.0	1.4	2.5
4	1.5	1.0	1.5
5	1.5	1.0	1.0

Table 7.9 Data for exercise 7.24

Decision	μ	$\bar{\theta}^0$	$\bar{\sigma}^0$
1	100	100	20
2	80	100	20
3	120	100	20
4	110	100	10
5	60	100	30

Table 7.10 Data for exercise 7.25

7.25 Repeat exercise 7.24 using the data in table 7.10, with $\sigma^\varepsilon = 10$.

7.26 Repeat exercise 7.24 using the data in table 7.11, with $\sigma^\varepsilon = 20$.

Decision	μ	$\bar{\theta}^0$	$\bar{\sigma}^0$
1	120	100	30
2	110	105	30
3	100	110	30
4	90	115	30
5	80	120	30

Table 7.11 Data for exercise 7.26

Theory questions

7.27 Assume that we have a standard normal prior about a true parameter μ which we assume is normally distributed with mean $\bar{\mu}^0$ and variance $(\sigma^0)^2$.

a) Given the observations $W^1, \ldots, W^n$, is $\bar{\mu}^n$ deterministic or random?

b) Given the observations $W^1, \ldots, W^n$, what is $\mathbb{E}(\mu|W^1, \ldots, W^n)$ (where μ is our truth)? Why is μ random given the first n experiments?

c) Given the observations $W^1, \ldots, W^n$, what is the mean and variance of $\bar{\mu}^{n+1}$? Why is $\bar{\mu}^{n+1}$ random?

7.28 What is the relationship between the deterministic regret $R^{static,\pi}$ (recall that this was done for a machine learning problem where the "decision" is to choose a parameter θ) in equation (7.86) and the regret $R^{\pi,n}(\omega)$ for a single sample path ω in equation (7.88)? Write the regret $R^{\pi,n}(\omega)$ in equation (7.88) in the context of a learning problem and explain what is meant by a sample ω.

7.29 What is the relationship between the expected regret $\mathbb{E}R^{\pi,n}$ in equation (7.89) and the pseudo-regret $\bar{R}^{\pi,n}$ in equation (7.89)? Is one always at least as large as the other? Describe a setting under which each would be appropriate.

Problem solving questions

7.30 There are seven alternatives with normally distributed priors on μ_x for $x \in \{1, 2, 3, 4, 5, 6, 7\}$ given in the table below:

Choice	$\bar{\mu}^n$	σ^n
1	5.0	9.0
2	3.0	8.0
3	5.0	10.0
4	4.5	12.0
5	5.0	8.0
6	5.5	6.0
7	4.0	8.0

Table 7.12 Priors

Without doing any calculations, state any relationships between the alternatives based on the knowledge gradient. For example, $1 < 2 < 3$ means 3 has a higher knowledge gradient than 2 which is better than 1 (if this was the case, you do not have to separately say that $1 < 3$).

7.31 Figure 7.16 shows the belief about an unknown function as three possible curves, where one of the three curves is the true function. Our goal is to find the point x^* that maximizes the function. Without doing any computation (or math), create a graph and draw the general shape of the knowledge gradient for each possible experiment x. [Hint: the knowledge gradient captures your ability to make a better decision using more information.]

7.32 Assume you are trying to find the best of five alternatives. The actual value μ_x, the initial estimate $\bar{\mu}_x^0$ and the initial standard deviation $\bar{\sigma}_x^0$ of each $\bar{\mu}_d^0$ are given in table 7.13. [This exercise does not require any numerical work.]

a) Consider the following learning policies:

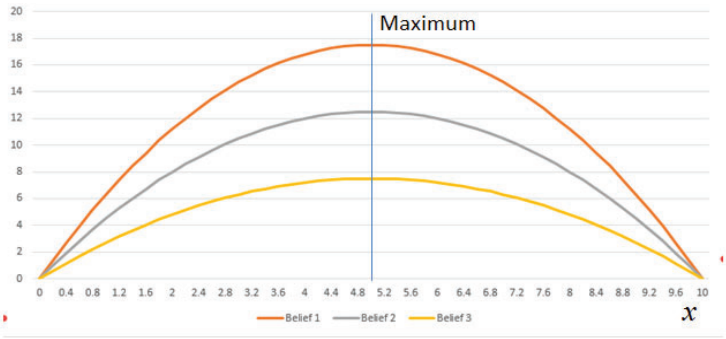


Figure 7.16 Use to plot the shape of the knowledge gradient for all x.

(1) Pure exploitation.

(2) Interval estimation.

(3) The upper confidence bounding (pick any variant).

(4) Thompson sampling.

(5) The knowledge gradient.

Write out each policy and identify any tunable parameters. How would you go about tuning the parameters?

b) Classify each of the policies above as a i) Policy function approximation (PFA), ii) Cost function approximation (CFA), iii) Policy based on a value function approximation (VFA), or iv) Direct lookahead approximation (DLA).

c) Set up the optimization formulation that can serve as a basis for evaluating these policies in an online (cumulative reward) setting (just one general formulation is needed - not one for each policy).

Alternative	μ	$\bar{\mu}^0$	$\bar{\sigma}^0$
1	1.4	1.0	2.5
2	1.2	1.2	2.5
3	1.0	1.4	2.5
4	1.5	1.0	1.5
5	1.5	1.0	1.0

Table 7.13 Prior beliefs for learning exercise

7.33 Joe Torre, former manager of the great Yankees, had to struggle with the constant game of guessing who his best hitters are. The problem is that he can only observe a hitter if he puts him in the order. He has four batters that he is looking at. The table below shows their actual batting averages (that is to say, batter 1 will produce hits 30 percent of the time, batter 2 will get hits 32 percent of the time, and so on). Unfortunately, Joe does not know these numbers. As far as he is concerned, these are all .300 hitters.

	Actual batting average			
	0.300	0.320	0.280	0.260
Day	Batter			
	A	B	C	D
1	0	1	1	1
2	1	0	0	0
3	0	0	0	0
4	1	1	1	1
5	1	1	0	0
6	0	0	0	0
7	0	0	1	0
8	1	0	0	0
9	0	1	0	0
10	0	1	0	1

Table 7.14 Data for problem 7.33

For each at-bat, Joe has to pick one of these hitters to hit. Table 7.14 below shows what would have happened if each batter were given a chance to hit (1 = hit, 0 = out). Again, Joe does not get to see all these numbers. He only gets to observe the outcome of the hitter who gets to hit.

Assume that Joe always lets the batter hit with the best batting average. Assume that he uses an initial batting average of .300 for each hitter (in case of a tie, use batter 1 over batter 2 over batter 3 over batter 4). Whenever a batter gets to hit, calculate a new batting average by putting an 80 percent weight on your previous estimate of his average plus a 20 percent weight on how he did for his at-bat. So, according to this logic, you would choose batter 1 first. Since he does not get a hit, his updated average would be $0.80(.200) + .20(0) = .240$. For the next at-bat, you would choose batter 2 because your estimate of his average is still .300, while your estimate for batter 1 is now .240.

After 10 at-bats, who would you conclude is your best batter? Comment on the limitations of this way of choosing the best batter. Do you have a better idea? (It would be nice if it were practical.)

7.34 In section 7.13.3, we showed for the transient learning problem that if M_t is the identity matrix, that the knowledge gradient for a transient truth was the same as the knowledge gradient for a stationary environment. Does this mean that the knowledge gradient produces the same behavior in both environments?

7.35 Describe the state variable S^n for a problem where $\mathcal{X} = \{x_1, \ldots, x_M\}$ is a set of discrete actions (also known as "arms") using a Bayesian belief model where $\bar{\mu}_x^n$ is the belief about alternative x and β_x^n is the precision. Now set up Bellman's equation and

characterize an optimal policy (assume we have a budget of N experiments) and answer the following:

a) What makes this equation so hard to solve?

b) What is different about the approach used for Gittins indices that makes this approach tractable? This approach requires a certain decomposition; how is the problem decomposed?

Sequential decision analytics and modeling

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

7.36 Read chapter 4, sections 4.1-4.4, on learning the best diabetes medication.

a) This is a sequential decision problem. What is the state variable?

b) Which of the four classes of policies are presented as a solution for this problem?

c) The problem of learning how a patient responds to different medications has to be resolved through field testing. What is the appropriate objective function for these problems?

d) The policy has a tunable parameter. Formulate the problem of tuning the parameter as a sequential decision problem. Assume that this is being done off-line in a simulator. Take care when formulating the objective function for optimizing the policy.

7.37 Read chapter 12, sections 12.1-12.4 (but only section 12.4.2), on ad-click optimization.

a) Section 12.4.2 presents an excitation policy. Which of the four classes of policies does this fall in?

b) The excitation policy has a tunable parameter ρ. One way to search for the best ρ is to discretize it to create a set of possible values $\{\rho_1, \rho_2, \ldots, \rho_K\}$. Describe belief models using:

   i) Independent beliefs.

   ii) Correlated beliefs.

   Describe a CFA policy for finding the best value of ρ within this set using either belief model.

7.38 Read chapter 12, sections 12.1-12.4 on ad-click optimization. We are going to focus on section 12.4.3 which proposes a knowledge gradient policy.

a) Describe in detail how to implement a knowledge gradient policy for this problem.

b) When observations are binary (the customer did or did not click on the ad), the noise in a single observation $W_{t+1,x}$ of ad x can be very noisy, which means the value of information from a single experiment can be quite low. A way to handle this is to use a lookahead model that looks forward τ time periods. Describe how to calculate the

knowledge gradient when looking forward τ time periods (instead of just one time period).

c) How would you go about selecting τ?

d) There are versions of the knowledge gradient for offline learning (maximizing final reward) and online learning (maximizing cumulative reward). Give the expressions for the knowledge gradient for both offline and online learning.

7.39 Continuing the exercise for chapter 4, assume that we have to tune the policy in the field rather than in the simulator. Model this problem as a sequential decision problem. Note that you will need a "policy" (some would call this an algorithm) for updating the tunable parameter θ that is separate from the policy for choosing the medication.

Diary problem

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

7.40 Pick one of the learning problems that arises in your diary problem, where you would need to respond adaptively to new information. Is the information process stationary or nonstationary? What discuss the pros and cons of:

a) A deterministic stepsize policy (identify which one you are considering).

b) A stochastic stepsize policy (identify which one you are considering).

c) An optimal stepsize policy (identify which one you are considering).

Bibliography

Ankenman, B., Nelson, B. L. & Staum, J. (2009), 'Stochastic Kriging for Simulation Metamodeling', *Operations Research* **58**(2), 371–382.

Auer, P., Cesa-bianchi, N. & Fischer, P. (2002), 'Finite-time analysis of the multiarmed bandit problem', *Machine Learning* **47**(2), 235–256.

Bechhofer, R. E., Santner, T. J. & Goldsman, D. M. (1995), *Design and analysis of experiments for statistical selection, screening, and multiple comparisons*, John Wiley & Sons, New York.

Berry, D. A. & Fristedt, B. (1985), *Bandit Problems*, Chapman and Hall, London.

Box, G. E. P. & Wilson, K. B. (1951), 'On the Experimental Attainment of Optimum Conditions', *Journal of the Royal Statistical Society Series B* **13**(1), 1–45.

Brezzi, M. & Lai, T. L. (2002), 'Optimal learning and experimentation in bandit problems', *Journal of Economic Dynamics and Control* **27**, 87–108.

Chang, H. S., Fu, M. C., Hu, J. & Marcus, S. I. (2007), *Simulation-based Algorithms for Markov Decision Processes*, Springer, Berlin.

Chen, C. H. (1995), An effective approach to smartly allocate computing budget for discrete event simulation, *in* '34th IEEE Conference on Decision and Control', Vol. 34, New Orleans, LA, pp. 2598–2603.

Chen, C. H., Yuan, Y., Chen, H. C., Yücesan, E. & Dai, L. (1998), Computing budget allocation for simulation experiments with different system structure, *in* 'Proceedings of the 30th conference on Winter simulation', pp. 735–742.

Chen, H. C., Chen, C. H., Dai, L. & Yucesan, E. (1997), A gradient approach for smartly allocating computing budget for discrete event simulation, *in* J. Charnes, D. Morrice, D. Brunner & J. Swain, eds, 'Proceedings of the 1996 Winter Simulation Conference', IEEE Press, Piscataway, NJ, USA, pp. 398–405.

Chen, H. C., Chen, C. H., Yucesan, E. & Yücesan, E. (2000), 'Computing efforts allocation for ordinal optimization and discrete event simulation', *IEEE Transactions on Automatic Control* **45**(5), 960–964.

Chick, S. E. & Gans, N. (2009), 'Economic analysis of simulation selection problems', *Management Science* **55**(3), 421–437.

Chick, S. E., K & Inoue, K. (2001), 'New two-stage and sequential procedures for selecting the best simulated system', *Operations Research* **49**(5), 732—-743.

DeGroot, M. H. (1970), *Optimal Statistical Decisions*, John Wiley and Sons.

Frazier, P. I. (2018), A Tutorial on Bayesian Optimization, Technical report, Cornell University, Ithaca NY.

Frazier, P. I. & Powell, W. B. (2010), 'Paradoxes in Learning and the Marginal Value of Information', *Decision Analysis* **7**(4), 378–403.

Frazier, P. I., Powell, W. B. & Dayanik, S. (2009), 'The Knowledge-Gradient Policy for Correlated Normal Beliefs', *INFORMS Journal on Computing* **21**(4), 599–613.

Frazier, P. I., Powell, W. B. & Dayanik, S. E. (2008), 'A knowledge-gradient policy for sequential information collection', *SIAM Journal on Control and Optimization* **47**(5), 2410–2439.

Fu, M. C. (2014), *Handbook of Simulation Optimization*, Springer, New York.

Fu, M. C., Hu, J.-Q., Chen, C.-H. & Xiong, X. (2007), 'Simulation Allocation for Determining the Best Design in the Presence of Correlated Sampling', *INFORMS J. on Computing* **19**, 101–111.

Gittins, J. (1979), 'Bandit processes and dynamic allocation indices', *Journal of the Royal Statistical Society. Series B (Methodological)* **41**(2), 148–177.

Gittins, J. (1981), 'Multiserver scheduling of jobs with increasing completion times', *Journal of Applied Probability* **16**, 321–324.

Gittins, J. (1989), 'Multi-armed Bandit Allocation Indices', *Wiley and Sons: New York*.

Gittins, J. & Jones, D. (1974), A dynamic allocation index for the sequential design of experiments, *in* J. Gani, ed., 'Progress in statistics', North Holland, Amsterdam, pp. 241—-266.

Gittins, J., Glazebrook, K. D. & Weber, R. R. (2011), *Multi-Armed Bandit Allocation Indices*, John Wiley & Sons, New York.

Gupta, S. S. & Miescke, K. J. (1996), 'Bayesian look ahead one-stage sampling allocations for selection of the best population', *Journal of statistical planning and inference* **54**(2), 229—-244.

He, D., Chick, S. E. & Chen, C.-h. (2007), 'Opportunity Cost and OCBA Selection Procedures in Ordinal Optimization for a Fixed Number of Alternative Systems', *IEEE Transactions on Systems Man and Cybernetics Part C-Applications and Reviews* **37**(5), 951–961.

Hong, J. & Nelson, B. L. (2006), 'Discrete Optimization via Simulation Using COMPASS', *Operations Research* **54**(1), 115–129.

Hong, L. & Nelson, B. L. (2007), 'A framework for locally convergent random-search algorithms for discrete optimization via simulation', *ACM Transactions on Modeling and Computer Simulation* **17**(4), 1–22.

Huang, D., Allen, T. T., Notz, W. I. & Zeng, N. (2006), 'Global Optimization of Stochastic Black-Box Systems via Sequential Kriging Meta-Models', *Journal of Global Optimization* **34**(3), 441–466.

Kaelbling, L. P. (1993), *Learning in embedded systems*, MIT Press, Cambridge, MA.

Kim, S.-h., Nelson, B. L. & Sciences, M. (2005), 'On the Asymptotic Validity of Fully Sequential Selection Procedures for Steady-State Simulation', *Industrial Engineering* pp. 1–37.

Lai, T. L. (1987), 'Adaptive Treatment Allocation and the Multi-Armed Bandit Problem', *Annals of Statistics* **15**(3), 1091–1114.

Lai, T. L. & Robbins, H. (1985), 'Asymptotically Efficient Adaptive Allocation Rules', *Advances in Applied Mathematics* **6**, 4–22.

Nelson, B. L. & Kim, S.-H. (2001), 'A fully sequential procedure for indifference-zone selection in simulation', *ACM Trans. Model. Comput. Simul.* **11**(3), 251–273.

Nelson, B. L., Swann, J., Goldsman, D. & Song, W. (2001), 'Simple procedures for selecting the best simulated system when the number of alternatives is large', *Operations Research* **49**, 950–963.

Powell, W. B. (2019), 'A unified framework for stochastic optimization', *European Journal of Operational Research* **275**(3), 795–821.

Powell, W. B. & Ryzhov, I. O. (2012), *Optimal Learning*, John Wiley & Sons, Hoboken, NJ.

Robbins, H. & Monro, S. (1951), 'A stochastic approximation method', *The Annals of Mathematical Statistics* **22**(3), 400–407.

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

Russo, D., Van Roy, B., Kazerouni, A., Osband, I. & Wen, Z. (2017), 'A Tutorial on Thompson Sampling', **11**(1), 1–96.

Ryzhov, I. O. & Powell, W. B. (2009), A Monte Carlo Knowledge Gradient Method for Learning Abatement Potential of Emissions Reduction Technologies, *in* M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin & R. G. Ingalls, eds, 'Proceedings of the 2009 Winter Simulation Conference', pp. 1492–1502.

Singh, S., Jaakkola, T., Littman, M. & Szepesvari, C. (2000), 'Convergence results for single-step on-policy reinforcement-learning algorithms', *Machine Learning* **38**(3), 287—-308.

Skinner, D. C. (1999), *Introduction to Decision Analysis*, Probabilistic Publishing, Gainesville, Fl.

Stein, M. L. (1999), *Interpolation of spatial data: Some theory for kriging*, Springer Verlag, New York.

Sutton, R. S. & Barto, A. G. (2018), *Reinforcement Learning: An Introduction*, 2nd edn, MIT Press, Cambridge, MA.

Thompson, W. R. (1933), 'On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of Two Samples', *Biometrika* **25**(3/4), 285–294.

Thrun, S. B. (1992), 'The role of exploration in learning control', *In White, D. A., & Sofge, D. A*.

Wang, Y., Wang, C., Powell, W. B. & Edu, P. P. (2016), The Knowledge Gradient for Sequential Decision Making with Stochastic Binary Feedbacks, *in* 'ICML-2016', Vol. 48, New York.

Weber, R. R. (1992), 'On the gittins index for multiarmed bandits', *The Annals of Applied Probability* **2**(4), 1024–1033.

Whittle, P. (1983), *Optimization Over Time: Dynamic Programming and Stochastic Control Volumes I and II*, Wiley Series in Probability and Statistics: Probability and Statistics, John Wiley & Sons, New York.

Yao, Y. (2006), Some results on the Gittins index for a normal reward process, *in* H. Ho, C. Ing & T. Lai, eds, 'Time Series and Related Topics: In Memory of Ching-Zong Wei', Institute of Mathematical Statistics, Beachwood, OH, USA, pp. 284–294.