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 19

DIRECT LOOKAHEAD POLICIES

Up to now we have considered three classes of policies: policy function approximations (PFAs), parametric cost function approximations (CFAs), and policies that depend on value function approximations (VFAs) which approximate the impact of a decision on the future through the state variable. All three of these policies depend on approximating some function, which means we are limited by our ability to create approximations that work well in practice.

Not surprisingly, we cannot always develop sufficiently accurate functional approximations. Policy function approximations have been most successful when decisions are simple decisions (think of buy low, sell high policies) or low-dimensional continuous controls that can be approximated using parametric or nonparametric functions (these might range from a linear function to a neural network). Cost function approximations require a deterministic model that provides a reasonable approximation. Value function approximations work well when the value function exhibits structure that can be exploited using the family of approximating architectures we presented in chapter 3 or chapter 18.

When all else fails (and it often does), we have to resort to direct lookahead policies (DLAs), which optimize over some horizon to help capture the impact of decisions made now on activities in the future, from which we can extract the decision we would make now. A few examples of problems which are likely going to require a full direct lookahead policy are:

EXAMPLE 19.1

Knowing whether to turn left or right at an intersection requires planning your path all the way to the destination.

EXAMPLE 19.2

Imagine a hurricane moving through a region, as often happens in the southeastern U.S. as well as southeast Asia. It is necessary to evacuate regions, but given the presence of network bottlenecks, regions need to be evacuated in a coordinated way. It is important to plan an evacuation for each zone z at each point in time to identify the zones where it is critical to evacuate now given the current state of the hurricane.

EXAMPLE 19.3

Knowing whether you can use your valuable supply of O-minus blood (which everyone can use) depends on the planned flow of donations and surgeries, as well as the age of your current O-minus inventories.

EXAMPLE 19.4

A financial planner planning for your retirement needs to estimate the risk that your portfolio might not hit a target for you to retire comfortably. This assessment will determine what investments to make now.

Direct lookahead policies represent a much more brute force approach for making decisions and, not surprisingly, are typically very hard computationally. As a result, the challenge here is introducing approximations that make this problem tractable. We divide these approximations into two broad categories:

Deterministic lookahead - This is the most common approach in practice when we need to use a direct lookahead policy. There are problems where this works quite well (think of navigation systems helping you find a good path over a dynamically varying transportation network), but there are settings where deterministic approximations simply will not work.

Stochastic lookahead - When we need a direct lookahead policy but where a deterministic lookahead would ignore critical issues, then we have to explore the world of stochastic lookahead models, which will be the focus of most of this chapter.

Our presentation is organized as follows:

Part I: Foundational material - This consists of the following general topics for DLA policies:

Section 19.1 - Creating the optimal direct lookahead policy. This serves as the foundation for any direct lookahead approximation.

Section 19.2 - We present the notation we use for our approximate lookahead model, and discuss the different approximation strategies.

Section 19.3 - We discuss the idea of using objectives for the lookahead model that are different than the base model.

Section 19.4 - We return to the familiar territory of evaluating a policy, a dimension that is often overlooked in the context of DLA policies.

Section 19.5 - We close Part I of this chapter with a discussion of why a DLA policy might be appropriate.

Part II: Deterministic DLA models - Section 19.6 discusses the simple but popular idea of using deterministic lookahead models.

Part III: Stochastic DLA models - We divide this substantial topic as follows:

Section 19.7 - We start with a quick tour through the four classes of policies, but discussed in the context of using them within a DLA model, where the tradeoff between computational cost and solution quality changes from when they are used in the base model:

Section 19.7.1 - Lookahead PFAs tend to be popular when available because they are computationally the easiest, but they are not the easiest to design.

Section 19.7.2 - Lookahead CFAs can be easier to tune if a deterministic approximation is available.

Section 19.7.3 - We describe strategies for using approximate dynamic programming, and particularly backward ADP, in the context of a lookahead model.

Section 19.7.4 - Using a DLA policy within a lookahead DLA model will be computationally demanding, but may be a necessary fallback.

Given this tour, we now present a few specialized results that have attracted considerable attention from different communities.

Section 19.8 - This is a thorough presentation of the popular idea of Monte Carlo tree search for problems with discrete actions. We cover both classical (pessimistic) MCTS and optimistic MCTS.

Section 19.9 - Next we cover two-stage stochastic programming which is widely used in the research literature for vector-valued decisions.

19.1 OPTIMAL POLICIES USING LOOKAHEAD MODELS

Direct lookahead policies are best described by restating our objective function

$$F(S_0) = V_0(S_0) = \max_{\pi \in \Pi} \mathbb{E}\left\{ \sum_{t'=0}^{T} C(S_{t'}, X_{t'}^{\pi}(S_{t'}))|S_0 \right\}. \tag{19.1}$$

We remind the reader of our habit of always conditioning the expectation on the starting state S_0; if you change the starting state, it can have an impact on the optimal policy. This means that we should technically be writing our optimal policy as a function of S_0, as in $\pi^*(S_0)$. Up to now we have typically left this dependence implicit, but as we progress, we will see that we need to remind ourselves of this dependence.

Now imagine solving equation (19.1) starting at time t,

$$V_t(S_t) = \max_{\pi \in \Pi} \mathbb{E}\left\{ \sum_{t'=t}^{T} C(S_{t'}, X_{t'}^{\pi}(S_{t'}))|S_t \right\}, \tag{19.2}$$

which we can also write as

$$V_t(S_t) = \max_{x_t \in \mathcal{X}_t} \Bigg(C(S_t, x_t) + \mathbb{E}_{S_t} \mathbb{E}_{W_{t+1}|S_t} \left\{ \max_{\pi \in \Pi} \mathbb{E}_{S_{t+1}} \mathbb{E}_{W_{t+1},\ldots,W_T|S_{t+1}} \left\{ \sum_{t'=t+1}^{T} C(S_{t'}, X_{t'}^{\pi}(S_{t'}))|S_{t+1} \right\} |S_t, x_t \right\} \Bigg), \tag{19.3}$$

where $\mathcal{X}_t$ are the constraints given the state S_t (which is implicit by indexing the constraint set by time t). We have written the expectations in (19.4) in the full expanded form. The expectations over S_t (or the imbedded S_{t+1}) would also handle any belief states. It is easy to see looking at (19.4) why we rarely use the expanded form, and instead we just write

$$V_t(S_t) = \max_{x_t \in \mathcal{X}_t} \left(C(S_t, x_t) + \mathbb{E} \left\{ \max_{\pi \in \Pi} \mathbb{E} \left\{ \sum_{t'=t+1}^{T} C(S_{t'}, X_{t'}^{\pi}(S_{t'}))|S_{t+1} \right\} |S_t, x_t \right\} \right), \tag{19.4}$$

We just caution the reader that when you see equation (19.4), we mean the expression in equation (19.3).

We can now write this as a policy for making the decision x_t at time t:

$$X_t^{DLA}(S_t) = \arg\max_{x_t} \left(C(S_t, x_t) + \mathbb{E} \left\{ \underbrace{\max_{\pi \in \Pi} \mathbb{E} \left\{ \sum_{t'=t+1}^{T} C(S_{t'}, X_{t'}^{\pi}(S_{t'}))|S_{t+1} \right\}}_{V_{t+1}(S_{t+1})} |S_t, x_t \right\} \right). \tag{19.5}$$

We note in passing that if we could compute (19.5), the policy $X_t^{DLA}(S_t)$ would be the optimal policy.

If we were able to compute the value function $V_{t+1}(S_{t+1})$ in equation (19.4), we would be able to write our policy as

$$X_t^{DLA}(S_t) = \arg\max_{x_t} \big(C(S_t, x_t) + \mathbb{E}\{V_{t+1}(S_{t+1})|S_t, x_t\} \big). \tag{19.6}$$

or, using the post-decision value function $V_t^x(S_t^x)$ which we introduced in chapter 15,

$$X_t^{DLA}(S_t) = \arg\max_{x_t} \left(C(S_t, x_t) + V_t^x(S_t^x) \right). \tag{19.7}$$

Remember that the optimization problem using the post-decision state is a deterministic problem (that is, no imbedded expectation), which opens the door to problems where x_t is a vector (possibly even a very high-dimensional vector), as long as we approximate $V_t^x(S_t^x)$ with a function that is linear or concave in x_t.

Of course, the versions of the policies using value functions are attractive because they are so compact, but if we could compute them, or even develop reasonable approximations, we would be drawing on the techniques we introduced in chapters 14 - 18. The reason we have direct lookahead policies is specifically for the many problems where value functions are simply not effective. Some examples of problems where the value of the future is not easy to approximate include:

- Problems with complex interactions - Imagine a stochastic scheduling problem (routing vehicles, scheduling machines, scheduling doctors) which involve complex interactions in the future between different types of resources (vehicles and loads, machines and jobs, doctors, nurses and patients). To make a decision now (for example, to commit to serve a job or patient in the future), it is necessary to explicitly plan the schedule in the future.

- Problems with forecasts - Consider a problem of managing inventories of products over a holiday, where we have a forecast $f_t = (f_{tt'})_{t' \geq t}$ for the demands. Since forecasts evolve over time, they should be a part of the state variable, but this is never done. The most natural approach is to model forecasts as latent variables (which means we ignore that the forecasts themselves are changing), but this requires that we model the problem over a planning horizon.

- Multilayer resource allocation problems - Value functions can be effective when modeling single layer resource allocation problems (managing water, blood, money, trucks), but many problems involve multiple layers (jobs and machines, trucks and packages, blood and patients). It is very hard to capture the value of machines (for example) when it depends on the number of jobs.

The policy in equation (19.5) can look daunting, but this is the starting point for the rest of this chapter. Figure 19.1 helps to explain it by drawing parallels with a decision tree which we first saw back in section 2.1.2. Figure 19.1 uses the convention that square nodes represent pre-decision states S_t (or $S_{t'}$ for $t' > t$) where decisions are made. Solid lines are decisions; dashed lines represent random outcomes, over which we have to take expectations. Keep in mind that we would never use a decision tree if, for example, x_t was a vector. In this chapter, we are going to keep open the possibility that x_t is a vector, which means the idea of enumerating all possible values of x_t is by itself intractable.

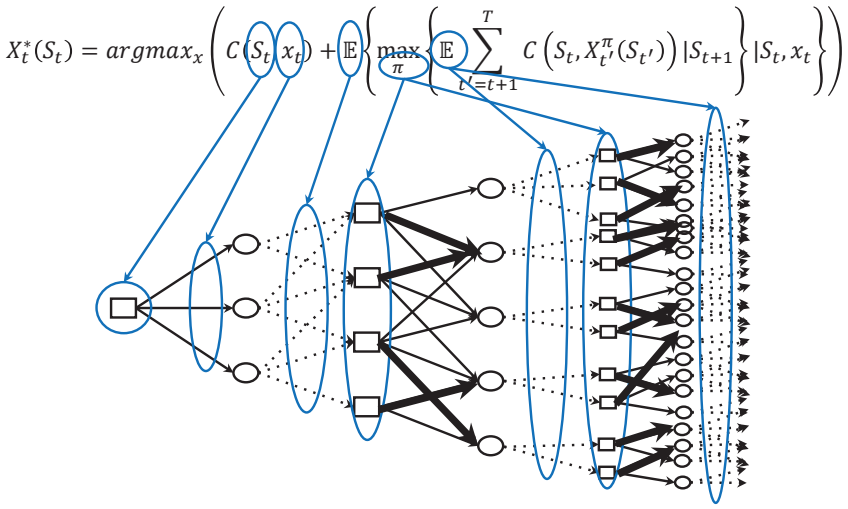

Figure 19.1 Relationship between the lookahead policy in equation (19.5) and a decision tree.

Whenever we see expectations, we generally assume that we cannot compute it since random variables can be continuous and/or vector-valued. However, Monte Carlo simulation is such a powerful tool, it is safe to assume that we are going to turn to Monte Carlo methods to approximate any expectation.

Figure 19.1 draws a line from the policy π that we are maximizing over to each decision node starting at time $t+1$. This reflects the property that the policy π in the lookahead model has to specify the decision for *each* decision node, from time $t+1$ onward. Designing this policy is easily the most difficult aspect of our lookahead policy in equation (19.5). We refer to this policy as the *lookahead policy*, but we also sometimes call it the "policy within a policy." Here, not only do we face the problem of searching over policies (a challenge we have been addressing since chapter 11), we have to do this *within the expectation*! This means we do this for *every* x_t and *every* outcome of W_{t+1} (more specifically, for every state S_{t+1})!! Needless to say we are going to need to design some shortcuts.

It should not be lost on us that the problem of finding a lookahead policy starting at time $t+1$ is really the same problem we are facing of designing a policy for time t (or time 0). All we have done is to push the problem out one time period. However, we also need to recognize that we are never going to actually implement the decisions produced by the lookahead policy; these are just to help us make a better decision now. We are going to exploit this observation by making the argument that we can better tolerate a suboptimal policy in the lookahead model than we can in the base model.

There are entire fields of research that pursue different strategies for approximating the lookahead model. For example:

Rolling horizon procedure - Also known as a "receding horizon procedure," it refers to the process of optimizing over an interval $(t, t+H)$, implementing decisions for time t, rolling to $t+1$ (and sampling/observing new information), and then solving over the interval $(t+1, t+H+1)$ (hence the name "rolling horizon"). Most of the time rolling horizon procedures use deterministic approximations of the future.

Model predictive control - This is the term used in the engineering-controls community, and refers to the fact that if we create a lookahead model, we need an explicit model of the problem, which means this is just another name for "direct lookahead policy." The controls literature in engineering focuses mostly on deterministic problems, and as a result, MPC (the standard abbreviation for model predictive control) is typically associated with deterministic models of the future, and MPC is most often associated with deterministic lookaheads.

Stochastic programming - The stochastic programming community simplifies the lookahead model in several ways, but the two most important are a) we represent future uncertainties with a sampled set of scenarios, and b) we simplify the future by assuming that we first see the entire future,and then optimize given that we are allowed to see the entire future (this is known as a "two-stage" approximation, reviewed below).

Monte Carlo tree search - MCTS is a an algorithmic strategy (reviewed below) for adaptively searching in a forward manner a stochastic search tree for problems with discrete action spaces. Asymptotically it will explore the entire search tree, which means it can be used to solve the base problem, but in practice it is a partial search, which means it is a direct lookahead policy.

Rollout policies - Here we assume that the lookahead policy is replaced with a simple function that is easy to compute, but technically this is just another word for any approximate policy.

Robust optimization - The field of robust optimization replaces the model of uncertainty with an "uncertainty set" where future outcomes $W_{t+1}, \ldots, W_{t+H}$ falls within a bounded region, and we then maximize $x_{t+1}, \ldots, x_{t+H}$ for a particular realization $w_{t+1}, \ldots, w_{t+H}$ which represents the worst outcome given the decisions.

Approximate dynamic programming - There are many applications of approximate dynamic programming which are actually implemented in the context of a stochastic lookahead model (but the authors forgot to tell you). Often, it is specifically the approximations that we are going to introduce below for creating approximate lookahead models that makes ADP possible.

Our entire presentation is centered on the idea of replacing the lookahead model with an approximation that is easier to solve, although we do present approaches that will solve the lookahead model to optimality, at least asymptotically. We begin by providing a notational system for modeling an approximate lookahead model, and then present different approximation strategies. The remainder of the chapter is then dedicated to describing these strategies in greater depth.

19.2 CREATING AN APPROXIMATE LOOKAHEAD MODEL

The art of direct lookahead policies is centered on replacing the true model (that is, our base model) with an approximate lookahead model that captures the most important aspects of our problem, while introducing simplifications that make the lookahead model tractable.

There are many research papers solving "stochastic, dynamic models" which are actually stochastic lookahead models with variables that are being held constant, which means they are not even discussed. These are not always obvious. In fact, whether a stochastic, dynamic model is a lookahead model or a base model often depends on how it is used. If we just want to implement the decision in the first period, after which new information arrives and the entire process repeats, it is a lookahead model. However, we might use our stochastic, dynamic model to test the effect of changes in input parameters over the entire horizon. In this case, the model is a base model being used as a simulator.

Two more examples illustrate this issue:

EXAMPLE 19.5

Brazil uses stochastic optimization models to plan their use of hydroelectric reservoirs. The plan optimizes over a 10 year period, and can be used in one of two ways. The first is to determine the flows of water between reservoirs over the upcoming week. This process is repeated each week. Used in this way, it is a stochastic lookahead model. However, the same model can be used to test changing the capacity of pumps for moving water between reservoirs. In this setting the model is used as a simulator (that is, a base model).

■ **EXAMPLE 19.6**

A stochastic model for optimizing the movement of trucks between locations can be used to determine how trucks should be dispatched. This model, which optimizes flows over a week, can then be updated every hour with new forecasts. Used in this way, it is a lookahead model. However, the same model can be used to simulate the effect of different fleet sizes, in which case the model is a base model used for strategic planning.

Below we introduce notation for the lookahead model, followed by a discussion of the different classes of approximations that we can introduce.

19.2.1 Modeling the lookahead model

We begin by observing that a DLA policy is solving a model, called the lookahead model, that is distinct from the base model that we are trying to solve by designing an effective policy. This means we need notation that is specific to the lookahead model.

We begin by noting that a lookahead model has to be indexed by the time t at which it is being formulated. Since it extends over a horizon from t to $\min\{t+H, T\}$, we index every variable by t (which fixes the information content of the model) and t', which is the time period within the lookahead horizon.

Then, we suggest using the same variables as in the base model, but with a "tilde." Thus, $\tilde{S}_{tt'}$ would be the state in our lookahead model at time t' within the lookahead horizon, for a model being solved at time t. $\tilde{S}_{tt'}$ might have fewer variables than S_t (or $S_{t'}$), and we might also use a different level of aggregation. Similarly, $\tilde{x}_{tt'}$ would be the decision we make at time t' using our lookahead policy $\tilde{\pi}$, given by the function $\tilde{X}^{\tilde{\pi}}_t(\tilde{S}_{tt'})$, with exogenous information $\widetilde{W}_{tt'}$ being the simulated information in our lookahead model that we first observe at time t'. With this notation, our sequence of states, decisions and "exogenous" information, for a lookahead model generated at time t, would look like

$$(\tilde{S}_{tt}, \tilde{x}_{tt}, \widetilde{W}_{t,t+1}, \tilde{S}_{t,t+1}, \tilde{x}_{t,t+1}, \widetilde{W}_{t,t+2}, \ldots, \tilde{S}_{tt'}, \tilde{x}_{tt'}, \widetilde{W}_{t,t'+1}, \ldots).$$

Note that in a base model, we may be running a process online which means that after we make a decision x_t at time t, the exogenous information W_{t+1} would be observed from a physical process. In a lookahead model, $\widetilde{W}_{t,t+1}$ must come from a model.

Using this notation, our direct lookahead policy would be written

$$X^{DLA}_t(S_t) = \arg\max_{x_t}\left(C(S_t,x_t) + \tilde{\mathbb{E}}\left\{\max_{\tilde{\pi}\in\tilde{\Pi}}\tilde{\mathbb{E}}\left\{\sum_{t'=t+1}^{t+H} C(\tilde{S}_{tt'}, \tilde{X}^{\tilde{\pi}}_{tt'}(\tilde{S}_{tt'}))|\tilde{S}_{t,t+1}\right\}|S_t,x_t\right\}\right) \tag{19.8}$$

where $\tilde{S}_{t,t'+1} = \tilde{S}^M(\tilde{S}_{tt'}, \tilde{X}^{\tilde{\pi}}_t(\tilde{S}_{tt'}), \widetilde{W}_{t,t'+1})$ describes the dynamics within our lookahead model, and where $\tilde{X}^{\tilde{\pi}}_t(\tilde{S}_{tt'})$ is the lookahead policy corresponding to $\tilde{\pi}$.

Here, we write $\tilde{\Pi}$ as a modified set of policies, and $\tilde{\mathbb{E}}$ as a the expectation over a modified set of random outcomes. In fact, later we are going to introduce the possibility of using a different uncertainty operator just for the lookahead model, such as one that evaluates extreme events to capture risk. We might even be modeling time differently (e.g. hourly time steps instead of 5 minutes), but we are going to keep the same time notation for simplicity.

19.2.2 Strategies for approximating the lookahead model

There are a variety of strategies that we can use for approximating the lookahead model to make solving (19.5) computationally tractable. These include:

1) Horizon truncation - We may reduce the horizon from (t, T) to $(t, t + H)$, where H is a suitable short horizon that is chosen to capture important behaviors. For example, we might want to model water reservoir management over a 10 year period, but a lookahead policy that extends one year (capturing a full cycle of seasons) might be enough to produce high quality decisions. We can then simulate our policy to produce forecasts of flows over all 10 years.

2) Outcome aggregation or sampling - Instead of using the full set of outcomes Ω (which is often infinite), we can use Monte Carlo sampling to choose a small set of possible outcomes that start at time t (assuming we are in state S_t^n during the nth simulation through the horizon) through the end of our horizon $t + H$. We refer to this as $\tilde{\Omega}_t^n$ to capture that it is constructed for the decision problem at time t while in state S_t^n. The simplest model in this class is a deterministic lookahead, which uses a single point estimate.

3) Discretization - Time, states, and decisions may all be discretized in a way that makes the resulting model computationally tractable. In some cases, this may result in a Markov decision process that may be solved exactly using backward dynamic programming (which we introduced in chapter 14). Because the discretization generally depends on the current state S_t, this model will have to be solved all over again after we make the transition from t to $t + 1$.

4) Stage aggregation - A stage represents the process of revealing information followed by the need to make a decision. We may approximate the future by aggregating stages to reduce the growth of the problem.

5) Latent variables - We may ignore some variables in our lookahead model as a form of simplification. For example, a forecast of weather or future prices can add a number of dimensions to the state variable (we demonstrated this in our energy storage example in section 9.9). While we have to track these in the base model (including the evolution of these forecasts), we can hold them fixed in the lookahead model, and then ignore them in the state variable (these become *latent variables*).

6) Policy approximation - The lookahead model still requires that we design a lookahead policy, which means we have to find a "policy-within-a-policy." While we may have chosen to use a lookahead policy for the base model, we would typically choose something simpler as the policy within the lookahead model.

The remainder of this chapter describes different strategies that have been used for approximating lookahead models. The most complex approximation strategy is the design of the lookahead policy for stochastic lookaheads, since this is where we have to recognize that while we are trying to design a policy for our original base model, using a stochastic lookahead model still requires that we solve a stochastic optimization problem starting at time $t + 1$ onward.

Below we provide a brief discussion of each of these strategies. The strategy of policy approximation is so rich that, beyond a brief sketch, we defer a more complete treatment of this to later in the chapter.

Horizon truncation

Horizon truncation is easily the simplest approximation that is almost always used in lookahead models. The general idea is to choose a horizon H that is long enough to capture activities in the future that we feel will affect decisions now. In an energy storage problem, this might require planning at least a day (sometimes two) into the future to capture daily cycles. In seasonal inventory planning, we might need a horizon that extends past peak periods such as a major holiday. A longer horizon usually means better results (but not always).

Outcome aggregation or sampling

We typically approximate the expectations in the lookahead model using samples. The first expectation is over the random variables in $\widetilde{W}_{t+1}$, while the second expectation is over entire sample paths of the sequence $\widetilde{W}_{t+2}, \widetilde{W}_{t+3}, \ldots, \widetilde{W}_{t+H}$. We might write the first set of samples as $\tilde{\Omega}_{t+1}$, while the second set of samples might be represented using the set $\tilde{\Omega}_{[t+2,t+H]}$.

As with the horizon, the more samples you have, the better your results will be, but the marginal improvement to the policy tends to decline while the computational cost increases. How much it increases depends on how you are making decisions. If we use a rollout policy, we have to repeat this for every sample, so the computational cost increases linearly with the size of the sample. However, there is a method called two-stage stochastic programming (described below) where we optimize over all scenarios, over all time periods, all at once. These problems can become quite large, and the CPU times can increase much faster than linearly with the sample size.

It is important to address the issue of sampling (which is always necessary) in conjunction with the design of the lookahead policy which we address below.

Stage aggregation

A common approximation is a two-stage formulation (see Figure 19.2(a)), where we make a decision x_t, then observe all future events (until $t + H$), and then make all remaining decisions. By contrast, a multistage formulation would explicitly model the sequence: decision, information, decision, information, and so on. Figure 19.2(b) illustrates the many possible paths in a multistage formulation, where we have highlighted a single history $h_{tt'}$ from t to t', followed by a set of outcomes after t' that share the common history $h_{tt'}$.

There is a substantial literature using the two-stage representation (figure 19.2(a)) for solving stochastic resource allocation problems, dating back to the 1950s. Section 19.9 provides a more detailed summary of this approach.

Latent variables

One of the most powerful, and subtle, approximation strategies involves simplifying the lookahead model by simply holding some variables constant, which means that they are treated as latent variables within the lookahead model. Some examples are:

- Forecasts - Easily one of the most common variables that are treated as latent variables is a forecast $f^W_{tt'}$ of a quantity $W_{t'}$ made at time t. As we step from time t to $t + 1$ in our base model, we would obtain updated forecasts $f^W_{t+1,t'}$. However, when we

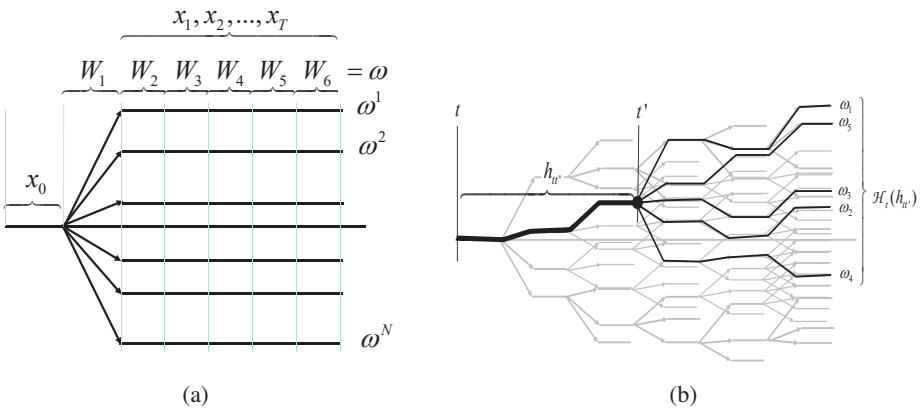

Figure 19.2 Illustration of (a) a two-stage scenario tree and (b) a multistage scenario tree showing a common history $h_{tt'}$ up to time t', with different sample paths after t'.

transition from time t' to $t'+1$ in the lookahead model, we do not update our vector of forecasts. As a result, the vector $(f^W_{tt'})_{t' \geq t}$ is held constant in the lookahead model, and as such are treated as latent variables.

- Beliefs - There are many settings where we have a belief B_t about a parameter, such as how a patient responds to a drug, such as one of two dozen drugs for reducing blood sugar. If we make a decision (such as prescribing medication) and then observe how the patient responds, then we learn from this response and update our belief. We may decide to try a medication to reduce blood sugar just to see how the patient responds. We then observe the response and update the belief. We may choose to make decisions in the lookahead model without updating beliefs, primarily because this can be computationally expensive. But we would update beliefs once we make our decision x_t and then observe the response W_{t+1} in the base model (which can be the field).

- We may be optimizing the movement of a utility truck cleaning up after a storm. Two forms of information are phone calls from customers complaining about power outages, and then the observations made by the utility truck observing where there are outages. To make the decision of how to route the truck, we certainly want to model the observations of the truck driver in the lookahead model, but we may wish to ignore new phone calls from customers in the lookahead model (remember that these would just be simulated phone calls).

There is no simple formula for deciding which dynamic state variables can be held fixed in the lookahead model. The key is to think about how the dynamic information would affect decisions in the future, and the extent to which this would affect the decision x_t that is implemented in the base model.

Policy approximation

The most challenging problem when creating a lookahead model is the design of the lookahead policy (or the policy-within-the-policy) given by equation (19.8). Just as most

of this book is on the design of good (ideally optimal) policies, the lookahead policy has its own optimization over policies imbedded within the lookahead. This sounds like a circular problem, since we are designing the lookahead policy as part of creating our DLA policy for the base model. A natural question is: if we have already decided we need a direct lookahead policy, shouldn't we just use a DLA policy (within our DLA policy)?

The first problem is that DLA policies (for stochastic lookahead models) can be computationally expensive, as this chapter will demonstrate. The lookahead DLA policy has to be computed many times, so using a DLA policy in the lookahead model may not be computationally feasible (although a deterministic lookahead is not out of the question). The good news is that since the lookahead policy is just being used to approximate decisions (we are not actually implementing the lookahead policy), we can get away with simpler policies that are "good enough" and computationally simpler.

This said, it is important to think about all four classes of policies when designing a lookahead policy. It can be worthwhile to flip back to our original discussion of how to design a policy in chapter 11 (in particular the discussion in section 11.10). The important differences between choosing a policy for the base model and choosing a policy within a lookahead model are:

- Computation - The policy within the DLA typically has to be simulated many times, so computational demands are much more important when designing a lookahead policy compared to the base model, where the policy only has to be computed once per time period.
- Effect of suboptimality - Suboptimality of a policy for the base model translates directly to decisions that are being implemented. By contrast, errors in a lookahead policy have a less direct impact on the decisions that are actually being implemented.

In other words, the emphasis in the lookahead policy shifts more to minimizing computational cost and less to creating the highest quality decisions.

We delve into each of the four classes in more depth below, but for now we make the following comments:

PFAs - Policy function approximations are easily the simplest to compute, but we have to find the function first, which involves its own estimation problem. Fitting a PFA is the most difficult among the four classes. Later we introduce some short cuts.

CFAs - Cost function approximations involve an imbedded optimization, which means it is capturing some problem structure. As a result, the tuning becomes easier, but the function will be more expensive to compute.

VFAs - There are inherent parallels between PFAs and VFAs, but VFAs tend to be easier to approximate (compared to PFAs and pure CFAs) since it is incorporating information about the downstream effect of a decision, whereas PFAs and CFAs have to slowly learn which parameter settings work best over time through repeated simulations.

DLAs - More than any of the other policy classes, DLAs are directly optimizing over a horizon, so they do the most to incorporate problem structure directly within the policy. Deterministic DLAS are, of course, relatively easy to compute (but as a rule, harder than any of the other three classes), but will typically involve tunable parameters to compensate for the errors introduced by the deterministic model. By contrast, stochastic DLAs are easily the hardest to compute of all the policies, but minimize tuning (which may still be required as a result of the approximations we introduce).

19.3 MODIFIED OBJECTIVES IN LOOKAHEAD MODELS

The discussion above described ways of creating approximate lookahead models, where the approximations are introduced explicitly to simplify the process of computing the lookahead model.

There are, however, reasons to change the lookahead model to create a policy with desired behaviors. We begin our discussion with the important special case of risk-adjusted policies, where we replace the expectation operator (which simply averages over outcomes) with a risk operator that puts different weights on specific events, typically related to the tails of distributions. We then discuss other types of modifications that we have encountered in our own work.

19.3.1 Managing risk

When uncertainty is involved (as it is throughout this volume), it is very common that some events are viewed as particularly undesirable. Some examples are

- Arriving late to an appointment.
- Not meeting the energy requirements of a system.
- Overdosing a patient, or using a drug that produces a bad reaction.
- Losing money on an investment, or losing an amount of money that exceeds the reserves (such as an insurance policy) that is used to cover these events.
- Losing a competition by choosing the wrong players.
- Running out of time before completing enough experiments to find the right material or design.

There is an extensive (and mathematically deep) field of research devoted to the handling of risk. We focus on the issues associated with modeling and computation. We begin by introducing an uncertainty operator $\rho(\cdot)$ that we use instead of an expectation that allows us to manipulate a sequence of contributions in any way. We follow this with three different ways of creating risk-adjusted behaviors, followed by a discussion on evaluating risk-adjusted policies.

Uncertainty operators and risk-adjusted performance

It is not unusual to need to incorporate risk into the design of a policy. There are several ways in which we can consider risk:

- Uncertainty in the contribution - We may be trying to maximize our total contribution (or minimize a cost) where we are worried about the possibility of low contributions.
- Risk that we violate a constraint, such as not arriving on time, or not meeting demand.
- Uncertainty in a separate performance metric - We may be trying to minimize the cost of delivering goods to customer, but we also wish to ensure that a driver gets home on time.

Concern about the uncertainty in the objective can be handled by replacing the expectation operator with a more general uncertainty operator $\rho(\cdot)$. We can define our operator $\rho(F_t^\pi)$ in different ways:

- Value at risk - We would use the distribution of F_t^π to compute the α-percentile, where we might use $\alpha = 0.10$ to capture the lower tail. We would write our uncertainty operator as $\rho_\alpha(F_t^\pi)$ to give us the α-percentile.

- Conditional value at risk - Widely known as CVar, which is also known under other names (such as average value at risk), CVar uses the average of the values $\hat{F}_t^\pi$ that fall below the α-percentile (assuming we are worried about low values).

- Risk adjusted performance - If $\bar{F}_t^\pi$ is the average of the outcomes and $\bar{\sigma}_t^\pi$ is the standard deviation, our risk adjusted performance might be

$$\rho(F_t^\pi|\theta) = \bar{F}_t^\pi - \theta\bar{\sigma}_t^\pi,$$

  where we might choose $\theta \approx 2$ to capture the lower tail.

- Intra-horizon deviations - Rather than focusing on aggregate statistics over the entire lookahead horizon, we may wish to focus on, say, the worst performance in an individual time period, or the number of times the contribution is above or below some target. In this case, we could write our risk metric using

$$\rho\big(C(\tilde{S}_{t,t+1}, \tilde{X}^\pi(\tilde{S}_{t,t+1})), C(\tilde{S}_{t,t+2}, \tilde{X}^\pi(\tilde{S}_{t,t+2})), \ldots, C(\tilde{S}_{t,t+H}, \tilde{X}^\pi(\tilde{S}_{t,t+H}))\big).$$

These risk metrics are all defined in terms of the variability of actual performance over different sample outcomes (often referred to as scenarios).

Typically we would use a modified contribution function such as

$$\bar{C}^{risk}(S_t, x_t) = \mathbb{E}_{W_{t+1}} C(S_t, x_t, W_{t+1}) + \eta\rho(S_t, x_t, W_{t+1})$$

which combines an expected contribution $\mathbb{E}_{W_{t+1}} C(S_t, x_t, W_{t+1})$ with our risk metric $\rho(S_t, x_t, W_{t+1})$ which we assume has access to current state information S_t, the decision x_t, as well as the random information (stock returns, patient performance) W_{t+1}. The parameter η governs the tradeoff between the importance of the expected contribution versus the outcome of tail events. Of course, we are applying this risk-adjusted contribution in the context of our lookahead model, which means we will use our lookahead state variable $\tilde{S}_{tt'}$, decision $\tilde{x}_{tt'} = \tilde{X}^\pi(\tilde{S}_{tt'}$, and our modeled exogenous information $\widetilde{W}_{t,t'+1}$.

For example, let the performance of the lookahead model following sample path $\tilde{\omega}$ be given by

$$\tilde{F}_t^\pi(S_t, x_t|\tilde{\omega}) = C(S_t, x_t) + \sum_{t'=t=1}^{t+H} C(\tilde{S}_{tt'}(\tilde{\omega}), \tilde{X}^\pi(\tilde{S}_{tt'}(\tilde{\omega})), \widetilde{W}_{t,t'+1}(\tilde{\omega}))$$

where $\tilde{S}_{t,t'+1}(\tilde{\omega}) = \tilde{S}^M(\tilde{S}_{tt'}(\tilde{\omega}), \tilde{x}_{tt'} = \tilde{X}^\pi(\tilde{S}_{tt'}(\tilde{\omega})), \widetilde{W}_{t,t'+1}(\tilde{\omega}))$ is the state at time $t'+1$ while following sample path $\tilde{\omega}$. $\tilde{F}_t^\pi(S_t, x_t|\tilde{\omega})$ is the total contribution from following lookahead policy $\tilde{X}^\pi(\tilde{S}_{tt'})$ along sample path $\tilde{\omega}$. We can then define $\tilde{F}_t^\pi(S_t, x_t)$ to be a random variable with outcomes $\tilde{F}_t^\pi(\tilde{\omega}_i)$ for sample path $\tilde{\omega}_i \in \tilde{\Omega}_{[t+2,t+H]}$.

We could, then, compute a risk-adjusted version of this total performance

$$\bar{F}_t^{risk-sum,\pi}(S_t,x_t) = \mathbb{E}\tilde{F}_t^{\pi} + \eta\rho(\tilde{F}_t^{\pi}). \tag{19.9}$$

Alternatively, we could have summed the risk-adjusted contributions

$$\begin{aligned}\bar{F}_t^{sum-risk,\pi}(S_t,x_t) &= C(S_t,x_t) + \sum_{t'=t+1}^{t+H} \mathbb{E}C(\tilde{S}_{tt'},\tilde{x}_{tt'},\widetilde{W}_{t,t+1}) + \\ &\quad \eta\rho(\tilde{S}_{tt'},\tilde{x}_{tt'},\widetilde{W}_{t,t+1}).\end{aligned} \tag{19.10}$$

If we were not considering risk, these two metrics would be the same, since the expectation of the sum of a set of random variables is equal to the sum of the expectations. This is not the case when we introduce risk measures. For this reason

$$\bar{F}_t^{risk-sum,\pi}(S_t,x_t) \neq \bar{F}_t^{sum-risk,\pi}(S_t,x_t),$$

and it is not even clear what the relationship would be. Even more important is that we are making decisions $\tilde{x}_{tt'} = \tilde{X}^{\pi}(\tilde{S}_{tt'})$ over time within the lookahead, so we have to consider the evaluation of the entire range of partial sums of contributions. The design of this lookahead policy has actually attracted considerable interest the research literature (see the bibliographic notes for some additional readings). This topic is beyond the scope of this book, but this discussion serves as a brief introduction.

To capture risk in a lookahead policy, it helps to describe the entire information flow in the lookahead model using

$$\tilde{h}^{\pi}_{[t,t']} = (\tilde{S}_{tt},\tilde{x}_{tt},\widetilde{W}_{t,t+1},\ldots,\tilde{S}_{t,t'}).$$

We call $\tilde{h}^{\pi}_{[t,t']}$ a "history" realizing that we are actually looking forward in time, and this is the history relative to time t' in the future. The history $\tilde{h}^{\pi}_{[t,t']}$ is a random vector which produces the realization $\tilde{h}^{\pi}_{[t,t']}(\omega)$ when following sample path ω, using policy π.

Given this history, we can compile any activity metrics we need. Let

$\rho^{LA}(\tilde{h}^{\pi}_{[t,t']}|\theta) =$ performance metric derived from the history $\tilde{h}^{\pi}_{[t,t']}$ in the lookahead model.

The risk operator $\rho^{LA}(h_{[t,t+H]}|\theta)$ can capture virtually any statistic from the future history, which is computed from the family of sample paths. So, just as an expectation averages across sample paths, the risk operator $\rho^{LA}(h_{[t,t+H]}|\theta)$ can be used to take a worst case, or the 10th or 90th percentile. The key is that we are simulating its value, so it is readily computable from a simulated lookahead model.

Risk-adjusted policies

Ultimately, however we handle risk, it is still just a lookahead policy, a topic that is easy to overlook. For example, we might create our risk-adjusted lookahead policy by using

$$X_t^{risk}(S_t|\theta) = \arg\max_{x_t\in\mathcal{X}_t} \bar{F}_t^{sum-risk,\pi}(S_t,x_t). \tag{19.11}$$

If x_t is discrete, which is to say that $\mathcal{X}_t = \{x_1,\ldots,x_M\}$, computing this policy is fairly straightforward. If it is continuous, and possibly vector-valued, we would need to draw on the tools of derivative-based (chapter 5) or derivative-free (chapter 7) stochastic search.

There is actually a wide range of applications where risk is an important issue. How risk is captured is highly problem dependent. Financial applications tend to work with tail deviations; energy applications will balance expected costs with penalties for outages; applications in health will focus on the possibility of bad health outcomes; design of buildings and bridges tend to emphasize the possibility of failure under extreme wind or earthquakes; inventory problems typically focus on avoiding running out of the resource being supplied; the control of electric vehicles might focus on the possibility of the battery running low.

Chance-constrained policies

The policy in equation (19.11) was designed to maximize (or minimize) a risk adjusted objective function. An alternative approach is to retain our original objective that we presented above (equation (19.5))

$$X_t^{CC}(S_t|\theta) = \arg\max_{x_t}\left(C(S_t,x_t) + \mathbb{E}\left\{\max_{\pi\in\Pi}\mathbb{E}\left\{\sum_{t'=t+1}^{T} C(S_{t'},X_{t'}^{\pi}(S_{t'}))|S_{t+1}\right\}|S_t,x_t\right\}\right) \tag{19.12}$$

but now we introduce a constraint that our risk operator $\rho^{LA}(\tilde{h}^{\pi}_{[t,t+H]})$ (which is a random variable since $\tilde{h}^{\pi}_{[t,t+H]}$ is random) meets some target θ^{target} with a desired probability P^{target}. This means we wish to maximize expected contributions over our lookahead model, subject to the probabilistic constraint

$$\mathbb{P}[\rho^{LA}(\tilde{h}^{\pi}_{[t,t+H]}) \leq \theta^{target}] \quad \geq \quad P^{target}. \tag{19.13}$$

The constraint (19.13) is known as a *chance constraint*. Although computing the probability $\mathbb{P}[\rho^{LA}(\tilde{h}^{\pi}_{[t,t+H]}) \leq \theta^{target}]$ can be challenging, a natural approach is to approximate it using a sample, just as we have been using to approximate expectations in the lookahead policy (or to evaluate the policy).

Robust optimization

Robust optimization first emerged as a way of solving engineering design problems that arise in areas such as civil and mechanical engineering. The problem is to design buildings subject to stresses from wind. The challenge is to find the least cost design (a minimization problem) under the worst wind condition, given by speed and direction. A mechanical engineer would face a similar problem designing the wing of a commercial aircraft, where the goal is to minimize weight while handling the most extreme stresses from wind.

Let $F(x,w)$ be the cost of design x given wind conditions w, where we might set $F(x,w)$ to a very large number of our design fails. Then let $\mathcal{W}$ be the space of "wind" outcomes, with element $w \in \mathcal{W}$. Let x capture all the design choices. We want to solve the optimization problem

$$\min_{x\in\mathcal{X}}\max_{w\in\mathcal{W}} F(x,w). \tag{19.14}$$

This formulation is known as a *robust optimization problem* since our solution x has to do well under the worst conditions. The feasible region $\mathcal{X}$ is understood to be governed by physical constraints that are well defined. By contrast, the design of the set $\mathcal{W}$ is not as

well defined, since we have to address the issue of the likelihood of extreme events. Let θ be a parameter that governs the likelihood of events that we want to include in $\mathcal{W}$. Then, we would write this set as $\mathcal{W}(\theta)$ to reflect the dependence of the set $\mathcal{W}$ on this parameter.

There are several strategies that we can use to construction $\mathcal{W}(\theta)$:

- Box constraints - We can represent $\mathcal{W}(\theta)$ as a set of constraints of the form

$$w^{min} \leq w \leq w^{max}.$$

  If w is a vector (say, wind speed and direction), we would have w^{min} and w^{max} for each dimension. These limits might be chosen so that the probability that each dimension falls outside of this range is θ.

  The problem with this strategy is that the model would likely pick the most extreme values of each dimension of w. So, we might have high wind, and separately a direction that produces the most stress. This logic would likely choose w that is the most extreme of both values.

- Joint constraints - Here we construct a region such that the probability that all dimensions fall outside of the boundary with some probability θ. Joint regions are harder to create and work with, especially when we adapt this idea to sequential problems.

This strategy has been adapted for fully sequential problems by formulating a direct lookahead model using the same principles, with the following changes:

- Replace x with $(\tilde{x}_{tt}, \tilde{x}_{t,t+1}, \ldots, \tilde{x}_{t,t+H})$.
- Let w be a specific realization of $(\widetilde{W}_{t,t+1}, \ldots, \widetilde{W}_{t,t+H})$, and let $\mathcal{W}(\theta)$ be the set of all possible values of w that we are willing to consider. We let $w = (\tilde{w}_{t,t+1}, \ldots, \tilde{w}_{t,t+H})$ be a specific sequence of realizations.

Switching to our more standard style of maximizing over x, our multiperiod robust optimization problem would be written

$$\max_{(\tilde{x}_{tt},\ldots,\tilde{x}_{t,t+H})\in\mathcal{X}} \quad \min_{(\tilde{w}_{tt},\ldots,\tilde{w}_{t,t+H})\in\mathcal{W}(\theta)} F(\tilde{x}_t, \tilde{w}_t). \tag{19.15}$$

Note that the optimization problem stated by (19.15) is a deterministic optimization problem. Our problem is to pick a single vector $\tilde{x}_t = (\tilde{x}_{tt}, \ldots, \tilde{x}_{t,t+H})$ and a single $\tilde{w}_t = (\tilde{w}_{tt}, \ldots, \tilde{w}_{t,t+H})$. Understanding that we are only interested in $\tilde{x}_{tt}$ (as is the case with all our direct lookahead policies), we can now write our robust optimization model as a policy

$$X_t^{RO}(S_t|\theta) = \arg \min_{(\tilde{x}_{tt},\ldots,\tilde{x}_{t,t+H})\in\mathcal{X}} \quad \max_{(\tilde{w}_{tt},\ldots,\tilde{w}_{t,t+H})\in\mathcal{W}(\theta)} F(\tilde{x}_t, \tilde{w}_t). \tag{19.16}$$

Although we are optimizing over the entire vector $(\tilde{x}_{tt}, \ldots, \tilde{x}_{t,t+H})$, the policy only implements $\tilde{x}_{tt}$.

Robust optimization has been promoted by some as a way of avoiding the need to create an underlying probability distribution, although this is a bit misleading. Creating an uncertainty set $\mathcal{W}(\theta)$ that produces desired behaviors is the most difficult challenge, both computationally (since the boundaries of $\mathcal{W}(\theta)$ have to consider the joint likelihood of all the random events) and in terms of modeling the underlying problem. Thus, while the

policy (19.16) does not have an explicit probability calculation, the underlying probability model is imbedded in the creation of $\mathcal{W}(\theta)$.

Often overlooked by the robust optimization community is that the formulation (19.16) is inherently "two-stage" and ignores the ability to make new decisions as information unfolds. How important this is depends on the problem setting.

19.3.2 Utility functions for multiobjective problems

There are many settings where we evaluate a policy using well-defined metrics such as profits, costs, or the time to complete a task. However, we need policies that understand that the real world is more complex. Some examples are:

- A ride hailing service or trucking company needs to minimize the miles that a driver moves empty, but the company also has to recognize that drivers need to get home on time.
- Similarly, while minimizing empty miles we want to make sure that customers are served on time.
- We wish to schedule machinery to maximize throughput, but machines have to be maintained, and it helps to have some slack in the schedule for maintenance or to absorb delays when a machine had to be stopped for mechanical reasons.
- We want to find the shortest path over a dynamic network, but without too many turns.
- We wish to optimize our bids in a market, but have to be careful not to become predictable to our competitors.

A simple and practical way of handling these issues is to introduce bonuses and penalties in the lookahead model, which are ignored when we are evaluating policy (which might occur in the field). The use of bonus and penalties to handle different objectives is a widely used heuristic. Often, these bonuses and penalties are introduced to find a solution that meets target performance metrics other than minimizing costs (for example), which means we are typically trying to meet a chance constraint. Tuning these bonuses and penalties closely parallels policy search, and uses the same tools.

19.3.3 Model discounting

When we introduced approximations in our lookahead model, it can make sense to discount decisions being simulated farther in the future under the rationale that the cumulative effect of our approximations make these decisions less important to the process of determining the best decision now. A discounted lookahead policy would be written

$$X_t^{DLA}(S_t) = \arg\max_{x_t}\left(C(S_t,x_t) + \tilde{\mathbb{E}}\left\{\max_{\tilde{\pi}\in\tilde{\Pi}}\tilde{\mathbb{E}}\left\{\sum_{t'=t+1}^{t+H}\lambda^{t'-t}C(\tilde{S}_{tt'},\tilde{X}^{\tilde{\pi}}_{tt'}(\tilde{S}_{tt'}))|\tilde{S}_{t,t+1}\right\}|S_t,x_t\right\}\right). \tag{19.17}$$

The parameter λ serves the same role as λ in TD(λ) (see section 16.1.4), in that it plays the role of an *algorithmic* discount factor.

19.4 EVALUATING DLA POLICIES

There is a surprisingly substantial tradition in the literature focusing on stochastic lookaheads to focus on solving the stochastic lookahead without recognizing that the solution is just the computation of a policy that still needs to be evaluated. Often, so much work is put into solving a stochastic lookahead policy, that the idea of simulating what is an otherwise cumbersome policy can seem impractical.

The methods for evaluating stochastic lookahead policies are the same we would use for any policy. There are two fundamental strategies we can use:

- Offline, using a simulator - This is the best way to do a comprehensive comparison of policies, including tuning policies within a class. This requires building a simulator, which can be a significant project in many settings such as energy, transportation and logistics, health, and finance. A simulator requires both a model of the dynamics of the physical system (which is captured in the transition function $S^M(S_t, x_t, W_{t+1})$) and a model of the exogenous information process $W_1, W_2, \ldots, W_t, \ldots$. There are two ways to simulate the information process:

  - Using a mathematical model - This requires creating mathematical models of the random variables in W_{t+1} given the current state S_t and most recent decision x_t. Mathematical models can be quite sophisticated (chapter 10 provides an introduction to a field known as uncertainty quantification), but provide the ability to perform repeated simulations, including simulations of physical processes that did not exist in history (such as major investments into wind and solar farms for energy generation).
  - Using history - This is easily the most common approach used in finance, where backtesting is a standard method. This involves creating sample paths of $W_1, W_2, \ldots, W_t, \ldots$ from different periods of history

  Offline simulators enjoy the advantages of controlled experiments, the ability to test in parallel, and in some cases the ability to approximate derivatives, opening the door to derivative-based stochastic search.

- Online, in the field - Simulators can be expensive to build (and have their own errors), so the alternative is often to observe how policies work in the field. Online evaluation means experimenting with a real physical system, which means that you have to actually live with the performance of a policy for a period of time. While field evaluations avoid modeling errors, they are slow (it takes a day to simulate a day), and there is no way to approximate derivatives. However, there are many settings where building a simulator is just not possible or appropriate, so this is not a strategy to be discarded.

We need to make the point that simulating policies can be quite noisy. While this is not universally true, we recommend that methods for policy evaluation and tuning be designed with the possibility that this may be true, and that some initial experiments be run to evaluate the level of uncertainty.

While we will note in closing that all policies need to be evaluated, simulating a PFA or CFA policy is fundamental to its effectiveness, while the same is not true of a carefully designed stochastic DLA policy.

19.4.1 Evaluating policies in a simulator

Assume (as we will do moving forward) that we are going to pick a lookahead policy $\tilde{X}_t^{\tilde{\pi}}(\tilde{S}_t)$ which may be from any of the four classes. This means we can write the policy in (19.8) without the imbedded $\max_{\tilde{\pi}}$ operator:

$$X_t^{DLA}(S_t|\theta) = \arg\max_{x_t}\left(C(S_t,x_t)+\tilde{\mathbb{E}}\left\{\tilde{\mathbb{E}}\left\{\sum_{t'=t+1}^{t+H} C(\tilde{S}_{tt'},\tilde{X}_{tt'}^{\tilde{\pi}}(\tilde{S}_{tt'}|\theta))|\tilde{S}_{t,t+1}\right\}|S_t,x_t\right\}\right) \tag{19.18}$$

where θ captures any parameters needed to control our policy (and by this, we mean the entire DLA policy, not just the lookahead policy $\tilde{X}_{tt'}^{\tilde{\pi}}(\tilde{S}_{tt'}|\theta)$). We approximate the first expectation in (19.18) using a set of samples of $\widetilde{W}_{t+1}$ that we represent using the set $\tilde{\Omega}_{t+1}$. We then approximate the second expectation by creating a series of sample paths $\omega \in \tilde{\Omega}_{[t+2,t+H]}$ where we use our pre-determined policy $\tilde{X}_t^{\tilde{\pi}}(\tilde{S}_t|\theta)$ to run a simulation along the sample path ω, creating the sequence

$$(\tilde{S}_{tt},\tilde{x}_{tt},\widetilde{W}_{t,t+1}(\omega),\tilde{S}_{t,t+1},\tilde{x}_{t,t+1},\widetilde{W}_{t,t+2}(\omega),\ldots,\tilde{S}_{tt'},\tilde{x}_{tt'},\widetilde{W}_{t,t'+1}(\omega),\ldots),$$

where $\tilde{x}_{tt'} = \tilde{X}^{\pi}(\tilde{S}_{tt'}|\theta)$ and where $\tilde{S}_{t,t'+1} = \tilde{S}^M(\tilde{S}_{tt'},\tilde{x}_{tt'},\widetilde{W}_{t,t'+1})$. From this sequence we can accumulate contributions from each decision using $C(\tilde{S}_{tt'},\tilde{x}_{tt'})$.

The steps of the procedure are given in figure 19.3 which returns an approximation $\bar{F}_t(x_t)$ for the performance of the policy starting at time t in state S_t.

The procedure in figure 19.3 can, of course, be used to produce the estimate

$$\bar{F}_0(x_0|S_0) \approx \mathbb{E}\{F(x^{\pi,N},W)|S_0\}.$$

We can think of $\bar{F}_0(x_0|S_0)$ as a sampled estimate of $\mathbb{E}\{F(x^{\pi,N},W)|S_0\}$, which can be used in any of our algorithms for derivative-free stochastic search. Since we are using a simulator, we would use the objective for offline learning given (in chapter 7) by

$$\max_{\pi}\mathbb{E}\{F(x^{\pi,N},W)|S_0\}.$$

Any of the four classes of policies may be used here. The choice of learning policy should be guided by the speed with which simulations can be run, and the variability from one run of the simulator to the next.

19.4.2 Evaluating risk-adjusted policies

Surprisingly, a common strategy for evaluating a risk-adjusted policy is to simulate it N times and take an average. Let $X_t^{RA}(S_t|\theta)$ be any of our risk-adjusted policies: $X_t^{risk}(S_t|\theta)$ (equation (19.11)), $X_t^{CC}(S_t|\theta)$ (equations (19.12) - (19.13)), or $X_t^{RO}(S_t|\theta)$ (equation (19.16)). One way to evaluate these policies is to simulate a sample path ω to obtain

$$F^{RA}(\omega|\theta) = \sum_{t=0}^{T} C(S_t(\omega),X^{RA}(S_t(\omega)|\theta)). \tag{19.19}$$

We might do several simulations and take an average

$$\bar{F}^{RA}(\theta) = \frac{1}{N}\sum_{n=1}^{N} F^{RA}(\omega^n|\theta), \tag{19.20}$$

Step 0. Initialization:

  Step 0a. Set initial state $\tilde{S}_t$.

  Step 0b. Choose the sample sets $\tilde{\Omega}_{t+1}$ for $\widetilde{W}_{t+1}$, and $\tilde{\Omega}_{[t+2,t+H]}$ for the entire sequence $\widetilde{W}_{t+2}, \ldots, \widetilde{W}_{t+H}$.

Step 1. Do for $x_t = x_1, x_2, \ldots, x_M$:

  Step 2. Compute $C_t(x_t) = C(\tilde{S}_t, x_t)$:

  Step 3. Do for $\tilde{w}_{t+1} = \{\tilde{w}_1, \tilde{w}_2, \ldots, \tilde{w}_M\} \in \tilde{\Omega}_{t+1}$:

    Step 4a. Find state $\tilde{S}_{t+1} = \tilde{S}^M(\tilde{S}_t, x_t, \tilde{w}_{t+1})$.

    Step 4b. Choose policy parameters $(\theta^{min}, \theta^{max})$ given $\tilde{S}_{t+1}$ [We don't actually do this in practice]

    Step 4c. Do for each sample path $\omega \in \tilde{\Omega}_{[t+2,t+H]}$: [These simulations should be done in parallel]

      Step 5a. Simulate $(\theta^{min}, \theta^{max})$ policy over $t' = t+1, t+2, \ldots, t+H$ [Works for any parameterized policy]

        Step 6a. Find $\tilde{x}_{t'} = \tilde{X}^\pi(\tilde{S}_{t'}|\theta^\pi = (s, S))$.

        Step 6b. Find contribution $C_{t'}(\tilde{x}_{t'}, \omega) = C(\tilde{S}_{t'}, \tilde{x}_{t'})$.

        Step 6c. Find $\widetilde{W}_{t'+1}(\omega)$.

        Step 6d. Find next state $\tilde{S}_{t'+1} = \tilde{S}^M(\tilde{S}_{t'}, \tilde{x}_{t'}, \widetilde{W}_{t'+1})$.

      Step 5b. Accumulate total contribution $F_t^\pi(x_t, \omega) = C_t(x_t) + \sum_{t'=t+1}^{t+H} C_{t'}(\tilde{x}_{t'}, \omega)$.

  Step 7. Find $\bar{F}_t^\pi(x_t|S_t) = \frac{1}{M}\sum_{i=1}^M F_t^\pi(x_t, \omega_i)$.

Step 8. Find $x_t^* = \arg\max_{x_t} \bar{F}(x_t|S_t)$ and return x_t^* and the function $\bar{F}_t^\pi(x_t|S_t)$.

Figure 19.3 Simulation of a lookahead policy.

which means we are evaluating our policy using a standard expectation-based metric, which we have written previously as

$$F^{RA}(\theta) = \mathbb{E}\left\{\sum_{t=0}^{T} C(S_t, X^{RA}(S_t|\theta))|S_0\right\}. \tag{19.21}$$

Now we just have to search for the best θ using our standard stochastic search methods.

The attentive reader might raise the question: we are using one objective in the lookahead model for the policy, and a different objective to evaluate the policy. Is this right? We first note that top professionals in the field have been known to do this (see the bibliographic notes for this section at the end of the chapter). Second, it can be argued that our risk-adjusted policy can be viewed just as we would any other policy. If it is executed frequently, it may make sense to evaluate its performance over time (which means taking an average).

This said, there is a logical inconsistency between using one objective in the lookahead model and a different objective for evaluating the policy. Think about making a decision at time 0: The lookahead model in a risk-adjusted policy is using one metric, and then we evaluate the policy using a different method. We can immediately create a policy that would work better by simply switching to an expectation-based lookahead model.

Evaluating a policy using risk metrics is fairly straightforward when using sampling-based methods. The operator $\rho^{LA}(h_{[t,t+H]}|\theta)$ is quite general, since virtually any metric can be calculated from a simulated history of contributions and other statistics. The only issue when evaluating a policy would arise when using a chance-constrained formulation,

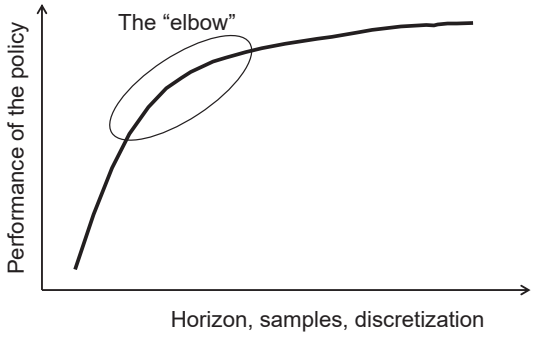

Figure 19.4 The performance elbow, which shows the improvement as we increase the horizon, number of samples or discretization intervals.

since we may find that we are evaluating the chance constraint. This would have to be handled by adding a penalty term, which is equivalent to dropping the chance constraint and moving these violations into the objective function.

19.4.3 Evaluating policies in the field

Evaluating a policy in the field means that we are going to be using the tools of derivative-free stochastic search, which we reviewed in chapter 7. In addition, it means that we have to use the objective which optimizes cumulative reward, given by

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

since we have to live with our performance while learning. Since learning is inherently quite slow, we suggest one of the lookahead policies in chapter 7 such as the knowledge gradient or one of its variants, which tend to be best suited for learning with limited budgets.

19.4.4 Tuning direct lookahead policies

The first three approximate strategies (the horizon, outcome sampling, and discretization intervals) are the most straightforward, since they fall into the "more is better" categories. More time periods (long horizons), more samples, and a finer discretization is always better. The problem is the tradeoff between solution quality and computational cost.

Finding the best value for each of these modeling choices tends to involve creating the plot depicted in figure 19.4, which illustrates the "performance elbow" which is the point where increasing the model parameter (horizon, outcomes, discretization intervals) produces diminishing returns. The point where CPU times become impractical depends entirely on the situation.

Whether we are searching over discrete choices (such as different classes of policies, or different types of approximation architectures), or continuous parameters, policy search is an exercise in stochastic search. Most frequently we are approaching the problem using the tools of derivative-free stochastic search (see chapter 7), but we may be able to use exact

or numerical derivatives for continuous parameters, and apply the methods of chapter 5. Either way, the discussion on policy search in chapter 12 is worth reviewing.

19.5 WHY USE A DLA?

Given the complexity of designing and computing a direct lookahead policy, it is fair to ask: what are the benefits? It is helpful to contrast two classes of policies for one of the most classical sequential decision problems: inventory planning. The simplest inventory problem features orders that arrive instantly, so the inventory equation is given by

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

where $\hat{D}_{t+1}$ is the demand arriving between t and $t+1$ that is not known when we place our ordering decision x_t (which arrives instantly). For this basic problem, our state variable is $S_t = R_t$. A natural policy (in fact, one that is known to be optimal for this problem) is a simple PFA known as an order-up-to policy written as

$$X^\pi(S_t|\theta) = \begin{cases} \theta^{max} - R_t & \text{if } R_t < \theta^{min}, \\ 0 & \text{otherwise.} \end{cases} \tag{19.22}$$

where we let $\theta = (\theta^{min}, \theta^{max})$.

This simple inventory problem is one of those cases where a reasonable policy is given by a known, analytical function. Now imagine that we are faced with the following series of variation that might easily arise for our inventory problem:

1) Start by introducing lead times, where an order x_t placed at time t arrives τ time periods later. In fact, imagine we are in North America ordering product from China, with a lead time of 100 days, although this can vary over a range of 50 days.

2) Next imagine that we recognize that at time 0, we have a ship enroute carrying an order from China that will arrive in 10 days.

3) Instead of one cargo ship arriving in 10 days, we have three ships arriving in 5, 15 and 40 days.

4) We are given information about a storm moving through the Pacific that could delay ships that intersect the path of the storm by 5 to 10 days. Alternatively, we have to plan for the event that a storm *might* arrive during the next 50 days, in which case it would delay our shipment.

5) Finally, we are given the option of placing a rush order via air freight that will arrive in 10 days.

We then pose the question: how do each of these variations change our decision?

Capturing these additional problem characteristics using an order-up-to policy is hard, because the only parameters we can control is the vector $\theta = (\theta^{min}, \theta^{max})$. This means we have to make θ a function of the state S_t, which we would write as $\theta(S_t)$. Information about ships arriving in the future and weather means that S_t is becoming fairly high dimensional (and remember, we are allowing ourselves to know that the starting policy is of the order-up-to variety). Even with this information, it would be quite difficult to design the function $\theta(S_t)$.

Now consider what happens when we use a direct lookahead. All of the information about ships arriving and weather (including uncertainty in the weather and the option of placing a rush order using air freight) would be captured in the lookahead model. As we simulate into the future, we would capture the ships arriving (this schedule is in the state variable), the random effect of the storm (this would be modeled in the random variables $\widetilde{W}_{tt'}$ in the lookahead model), and the ability to place a rush order using air freight (this would be a decision $\tilde{x}_{tt'}$ in the lookahead model, determined by some reasonable lookahead policy). Of course, this simulation has to be run for each ordering decision x_t now, but remember that all this can be done in parallel.

So, while we pay a computational cost for using a stochastic direct lookahead policy, we obtain a policy that not only captures all the dimensions of a very complex state variable S_t, we can also capture our ability to make decisions in the future. If we change the schedule of inbound ships, this may change our order decision x_t now. Similarly, if we introduce new decisions that we may make in the future in our lookahead model that help us respond to random events, that may change which x_t now is the best decision.

In other words, our DLA policy captures our highly complex state variable (the schedule of ships arriving), random future events (such as the storm), as well as options that help us to mitigate these random events (such as the rush order of inventory). This is a very responsive policy, without the need to do sophisticated machine learning.

This finishes Part I of this chapter, where we have discussed the general setting of DLA policies and the different strategies for overcoming the computational issues inherent in stochastic lookahead models. We next turn our attention first to the use of deterministic lookahead policies, which are easily the most popular class of policies for solving lookahead problems. Then, the rest of the chapter covers methods for approximating stochastic lookahead policies.

19.6 DETERMINISTIC LOOKAHEADS

The most widely used approximation used when designing a lookahead model is to assume that the problem is deterministic. This eliminates both expectations, which then means that instead of optimizing over policies, we just optimize over the decisions as with any deterministic model. The policy would then be written

$$\begin{aligned} X_t^{DLA-Det}(S_t) &= \arg\max_{x_t}\left(C(S_t,x_t)+\max_{x_{t+1},\ldots,x_{t+H}}\sum_{t'=t+1}^{T}C(S_{t'},x_{t'})\right),\\ &= \arg\max_{x_t,\ldots,x_{t+H}}\left(C(S_t,x_t)+\sum_{t'=t+1}^{T}C(S_{t'},x_{t'})\right),\\ &= \arg\max_{x_t,\ldots,x_{t+H}}\sum_{t'=t}^{T}C(S_{t'},x_{t'}). \end{aligned} \tag{19.23}$$

subject to the necessary constraints.

This is one policy where it does not matter whether x_t is a scalar (continuous or discrete) or a vector. Even if x_t is a scalar, the optimization problem in (19.23) still has to optimize over the vector $x_t,\ldots,x_{t+H}$.

There are a few myths that we have to address about deterministic lookahead policies:

Myth 1 - *Deterministic models are easy to solve.* While this can be true, there are complex problems where optimizing over long horizons can be very expensive. For example, optimizing a fleet of truck drivers or locomotives over horizons of a week or more can be computationally completely intractable. Below we are going to suggest stochastic lookahead models that are easier to solve than a deterministic lookahead for some problems.

Figure 19.5 shows the increase in CPU times for a transportation application that involves managing locomotives over time. If we increase the horizon to four days, which is not unreasonable in this setting, the CPU time to solve a single instance of a lookahead model grows to 50 hours (this work was done around year 2010 using Cplex and a large memory computer).

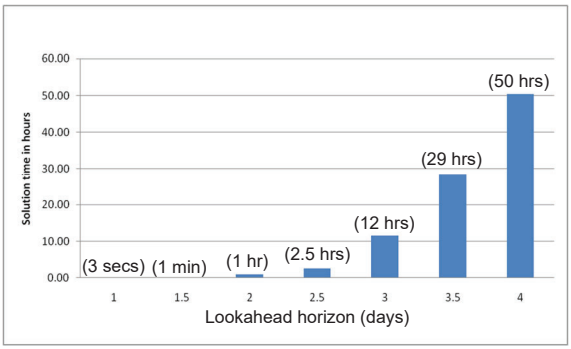

Figure 19.5 Growth in CPU times as we increase the horizon in a deterministic lookahead model.

Myth 2 - *Since we are obtaining an optimal solution, "it" is optimal.* This is a surprisingly common misconception. Often people do not appreciate that optimizing over a horizon to determine what to do now is a policy to solve a problem, not the problem itself, and an optimal solution to an approximate lookahead model is *not* an optimal policy! For example, think about planning a path using a navigation system, which finds the best path given point estimates of the time required to traverse each link of the network. This is likely to be a good policy, but because we are ignoring the uncertainty in those link times, this is not an optimal policy.

Myth 3 - *Even though the policy may not be optimal, it is a good starting point.* While there are many problems for which this is true, imagine using a deterministic lookahead to create a policy for buying and selling assets in the face of stochastically varying prices. The best estimate of future prices is the current price, so the forecast of future prices is constant. A natural policy is to buy when the price goes below some point, and sell when it goes above another point. A deterministic lookahead will not even produce a reasonable approximation of this policy. This said, it is entirely possible that a deterministic lookahead *may* be a good starting point, as we demonstrated in chapter 13 when we used a deterministic lookahead for our energy storage problem, with parametric modifications that had to be tuned.

We would still say that a deterministic lookahead should be considered as a possible starting point, but given the diversity of application settings, you have to think about whether a

deterministic lookahead would capture the right behaviors for your problem. You cannot just adopt a deterministic lookahead (or any policy) blindly.

19.6.1 A deterministic lookahead: shortest path problems

The best way to illustrate a lookahead policy is the process of finding the best path over a transportation network where travel times are evolving randomly as traffic moves. Imagine that we are trying to get from an origin q to a destination r, and assume that we are at an intermediate node i (trying to get to r). Our navigation system will recommend that we go from i to some node j by first finding the shortest path from i to r, and then using this path to determine what to do now.

This problem is solved as a deterministic (but time-dependent) dynamic programming problem. To simplify the notation, we are going to assume that each movement over a link (i, j) takes one time period. We represent our traveler only when he is at a node, since this is the only time when there is a real decision. Imagine that it is time t, and that we are at node q heading to r. Define

$$\begin{aligned} c_{tij} &= \text{the estimated cost, made at time } t, \text{ of traversing link } (i,j), \\ x_{tij} &= \text{the flow that we plan, at time } t, \text{ on traversing link } (i,j) \text{ (typically at some time in the future).} \end{aligned}$$

In our shortest path problem, the flow x_{tij} is either 1, meaning that link (i, j) is in the shortest path from q to r, and 0 otherwise.

Assume that we are sure we can arrive by time T, but if we arrive earlier, then we do nothing at node r until time T. We can write our problem as

$$\min_{x_t,\ldots,x_T} \sum_{t'=t}^{T} \sum_{i} \sum_{j} c_{tij} x_{tij} \tag{19.24}$$

subject to flow conservation constraints:

$$\sum_{j} x_{tqj} = 1, \tag{19.25}$$

$$\sum_{k} x_{t',ki} - \sum_{j} x_{t'+1,ij} = 0, \ t' = t, \ldots, T-1, \ \forall i, \tag{19.26}$$

$$\sum_{i} x_{T-1,ir} = 1. \tag{19.27}$$

The optimization model (19.24)-(19.27) is a lookahead model that is optimizing the problem from time t until the end of the horizon T. Constraint (19.25) specifies that one unit of flow has to go out of origin node q at time t. Constraints (19.26) ensure that flow into each intermediate node is equal to the flow out. Finally, constraint (19.27) ensures that there is one unit of flow into node r at time T.

Shortest path problems are always solved as (highly specialized) deterministic dynamic programs. When combined with some careful software engineering, problem (19.24)-(19.27) can be easily solved using Bellman's equation for deterministic problems:

$$V_{t'i} = \min_{j} \left(c_{t'ij} + V_{t'+1,j} \right), \tag{19.28}$$

for $t' = t, \ldots, T$ and all nodes i.

Both the linear program (19.24)-(19.27) and the deterministic dynamic program (19.28) represent deterministic lookahead models. When we solve the linear program, all we use is the decision x_t that tells us what to do at time t. Similarly, we use the decision from our dynamic program

$$x_t^* = \arg\min_j \left(c_{tqj} + V_{t+1,j}\right)$$

which tells us which node j we should go to. If $x_{tqj} = 1$, we will reoptimize when we arrive at node j at time $t + 1$, at which time the costs may have changed.

This problem is a nice illustration of a complex stochastic network problem where the state variable for the original problem is not just the location of the traveler, but the current estimate of the travel times on each link in the entire network. Needless to say, this is an exceptionally high-dimensional stochastic optimization problem for which there is no known algorithm that could solve it to optimality. By virtue of our choice of a deterministic lookahead model, we can solve the lookahead model optimally as a dynamic program, but this does not make it an optimal policy.

Shortest path problems represents a familiar application (we use this any time we use a navigation system) where a deterministic lookahead policy seems to provide useful guidance. If only all problems were this easy, as we demonstrate in the next section.

19.6.2 Parameterized lookaheads

One of the most powerful strategies for designing DLAs is to use a parameterized deterministic lookahead, as we did in section 13.3.3 for a stochastic, time-dependent energy storage problem. Given how important this strategy is for DLA policies, we are going to briefly summarize this strategy here.

Recall that we were optimizing over the following flows connecting a wind farm, the grid, a battery storage device, all serving a demand:

$$\begin{aligned}
\tilde{x}_{tt'} &= \text{Planned generation of energy during hour } t' > t, \text{ where the plan is made at time } t, \text{ which is comprised of the following elements:} \\
\tilde{x}^{ED}_{tt'} &= \text{flow of energy from renewables to demand,} \\
\tilde{x}^{EB}_{tt'} &= \text{flow of energy from renewables to battery,} \\
\tilde{x}^{GD}_{tt'} &= \text{flow of energy from grid to demand,} \\
\tilde{x}^{GB}_{tt'} &= \text{flow of energy from grid to battery,} \\
\tilde{x}^{BD}_{tt'} &= \text{flow of energy from battery to demand.}
\end{aligned}$$

The goal was to optimize the available energy E_t from the wind farm, energy from the grid that can be purchased at price p_t, to serve a demand D_t. Given the time-varying nature of demands, capacity constraints on storage and transmission, and the highly stochastic nature of wind energy, we need to plan into the future, where we use forecasts of demands D_t and energy from the wind farm E_t that we represent using

$$\begin{aligned}
f^D_{tt'} &= \text{forecast of demand } D_{t'} \text{ made at time } t, \\
f^E_{tt'} &= \text{forecast of wind energy } E_{t'} \text{ made at time } t.
\end{aligned}$$

We then created a deterministic lookahead policy where we optimize over a horizon $t, \ldots, t+H$ which is given by

$$X^{DLA}(S_t|\theta) = \arg \max_{x_t, (\tilde{x}_{tt'}, t'=t+1,\ldots,t+H)} \left(p_t(x_t^{GB} + x_t^{GD}) + \sum_{t'=t+1}^{t+H} \tilde{p}_{tt'}(\tilde{x}_{tt'}^{GB} + \tilde{x}_{tt'}^{GD}) \right) \tag{19.29}$$

subject to the following constraints: First, for time t we have:

$$\begin{aligned}
x_t^{BD} - x_t^{GB} - x_t^{EB} &\leq R_t, && (19.30)\\
\tilde{R}_{t,t+1} - (x_t^{GB} + x_t^{EB} - x_t^{BD}) &= R_t, && (19.31)\\
x_t^{ED} + x_t^{BD} + x_t^{GD} &= D_t, && (19.32)\\
x_t^{EB} + x_t^{ED} &\leq E_t, && (19.33)\\
x_t^{GD}, x_t^{EB}, x_t^{ED}, x_t^{BD} &\geq 0. && (19.34)
\end{aligned}$$

Then, for $t' = t+1, \ldots, t+H$ we have:

$$\begin{aligned}
\tilde{x}_{tt'}^{BD} - \tilde{x}_{tt'}^{GB} - \tilde{x}_{tt'}^{EB} &\leq \tilde{R}_{tt'}, && (19.35)\\
\tilde{R}_{t,t'+1} - (\tilde{x}_{tt'}^{GB} + \tilde{x}_{tt'}^{EB} - \tilde{x}_{tt'}^{BD}) &= \tilde{R}_{tt'}, && (19.36)\\
\tilde{x}_{tt'}^{ED} + \tilde{x}_{tt'}^{BD} + \tilde{x}_{tt'}^{GD} &= \theta_{t'-t}^{D} f_{tt'}^{D}, && (19.37)\\
\tilde{x}_{tt'}^{EB} + \tilde{x}_{tt'}^{ED} &\leq \theta_{t'-t}^{E} f_{tt'}^{E}. && (19.38)
\end{aligned}$$

The two key constraints are equations (19.37) and (19.38) which use the forecasts $f_{tt'}^{D}$ and $f_{tt'}^{E}$. Given the uncertainty in these forecasts, we multiply both by coefficients $\theta_{t'-t}^{D}$ and $\theta_{t'-t}^{E}$. These coefficients give us a parameterized policy $X^{DLA}(S_t|\theta)$, where we now face the challenge of tuning θ. We now have to tune θ by optimizing

$$\max_{\theta} F^{\pi}(\theta) = \mathbb{E}\left\{ \sum_{t=0}^{T} C(S_t, X^{\pi}(S_t|\theta))|S_0 \right\}. \tag{19.39}$$

A detailed summary of the tuning process for this problem is given in section 13.3.3. This application brings out the general strategy of using the concept of parametric cost function approximations in the setting of deterministic lookaheads. It is our belief that this strategy is actually widely used in practice, but in a ad hoc manner. What is missing is the formal process of tuning the parameters, which means solving the optimization problem given by (19.39), whether it is done in an online or offline setting.

We emphasize that no strategy is a panacea, but tuned deterministic lookaheads represent a practical and powerful strategy that has been largely ignored by the academic community. We note that one major strength of this approach over stochastic lookaheads is that we can tune θ in a very realistic simulator (or the real world), avoiding the array of approximations that are typically needed for stochastic lookaheads, as we describe next.

As with any parametric model, designing the parameterization is an art that requires some intuition into the structure of the problem and how uncertainty would affect a deterministic solution. We encourage readers to return to chapter 13 for the presentation on parameterized deterministic models.

19.7 A TOUR OF STOCHASTIC LOOKAHEAD POLICIES

We are now going to review each of the four classes of policies as potential candidates for the lookahead policy. While this tour parallels the path of this entire book that focuses on covering all four classes of policies, when viewed as candidates for a lookahead policy inside a direct lookahead policy shifts the emphasis to computation, with relaxed emphasis on solution quality.

As of this writing, there is very little research addressing the effect of suboptimal lookahead policies on the performance of a DLA policy. While there is some theoretical research, we anticipate that this evaluation is always going to be problem dependent, and will require testing in a simulator.

19.7.1 Lookahead PFAs

Our lookahead policy $X_t^{DLA}(S_t)$, with the imbedded lookahead policy $\tilde{X}_{t'}^{\tilde{\pi}}(\tilde{S}_{tt'})$, is given by

$$X_t^{DLA}(S_t) = \arg\max_{x_t} \left(C(S_t, x_t) + \tilde{E} \left\{ \max_{\tilde{\pi} \in \tilde{\Pi}} \tilde{E} \left\{ \sum_{t'=t+1}^{T} C(S_{t'}, \tilde{X}_{t'}^{\tilde{\pi}}(\tilde{S}_{tt'})) | \tilde{S}_{t+1} \right\} | S_t, x_t \right\} \right). \tag{19.40}$$

To break this down, assume for illustration that we are solving an inventory replenishment problem, and that our lookahead policy $\tilde{X}_{t'}^{\tilde{\pi}}(\tilde{S}_{tt'})$ follows an order-up-to policy (see equation (19.22)), where we trigger an order when the inventory R_t falls below θ^{min}, at which time we order $\tilde{X}_{t'}^{\tilde{\pi}}(\tilde{S}_{tt'}|\tilde{\theta}) = (\theta^{max} - R_t)$. Let $\tilde{\theta} = (\theta^{min}, \theta^{max})$ be our tunable parameters. If we fix this as our lookahead policy, we would replace the inner $\max_{\tilde{\pi}}$ with $\max_{\tilde{\theta}}$, giving us

$$X_t^{DLA}(S_t) = \arg\max_{x_t} \left(C(S_t, x_t) + \tilde{E} \left\{ \max_{\tilde{\theta}} \tilde{E} \left\{ \sum_{t'=t+1}^{T} C(S_{t'}, \tilde{X}_{t'}^{\tilde{\pi}}(\tilde{S}_{tt'}|\tilde{\theta})) | \tilde{S}_{t+1} \right\} | S_t, x_t \right\} \right). \tag{19.41}$$

Written this way, we see that the optimal $\tilde{\theta}^*$ depends on the state S_{t+1}, just as our optimal policy $X^*(S_t)$ from solving equation (19.1) should be written $X^*(S_t|S_0)$. We already discussed the challenges of creating a function $\theta^*(S_t)$ in section 19.5. We either need a general function $\theta^*(S_t)$ so we can compute $\tilde{\theta} = \theta^*(S_{t+1})$, or we have to find the optimal $\tilde{\theta}$ for each state S_{t+1} in (19.41). To be completely honest, neither of these seem feasible.

A more realistic alternative is that we are going to pick a single $\theta = (\theta^{min}, \theta^{max})$ for our DLA policy $X_t^{DLA}(S_t|\theta)$. Now our policy looks like

$$X_t^{DLA}(S_t|\theta) = \arg\max_{x_t} \left(C(S_t, x_t) + \tilde{E} \left\{ \tilde{E} \left\{ \sum_{t'=t+1}^{T} C(S_{t'}, \tilde{X}_{t'}^{\tilde{\pi}}(\tilde{S}_{tt'}|\tilde{\theta})) | \tilde{S}_{t+1} \right\} | S_t, x_t \right\} \right).$$

We still have to tune θ just as we would tune any parameter in a policy (as we have done for PFAs and CFAs), but now this only has to be done once (offline), which makes our policy much easier to compute.

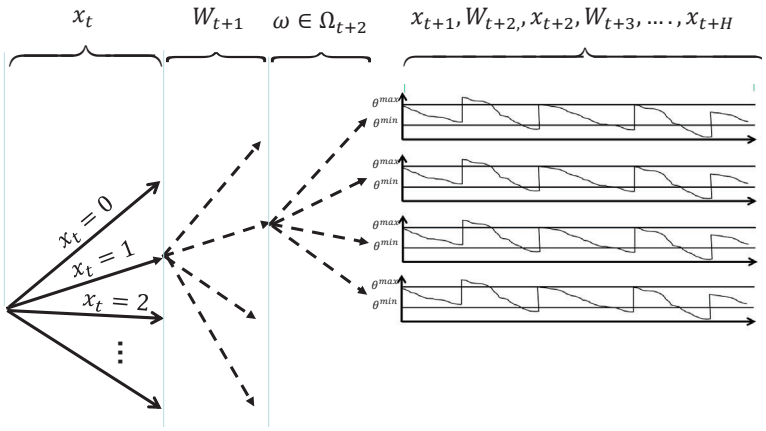

Figure 19.6 Illustration of the simulation of a direct lookahead inventory policy with an imbedded order-up-to policy within the lookahead model.

Of course, we have traded off solution quality for computational simplicity, but this is an example of prioritizing computation in a DLA policy, recognizing that a suboptimal policy does not have as much of an adverse impact since we are not actually implementing the decisions.

Figure 19.6 illustrates the calculations behind our lookahead policy in equation (19.41). We first have to loop over the possible values of x_t (so, x_t better be a discrete scalar). We then have to approximate the expectation over the random variable $\widetilde{W}_{t+1}$ using a Monte Carlo sample. Then, we create a series of sample paths, where one sample path ω is a sequence of observations of $\widetilde{W}_{t+2}, \widetilde{W}_{t+3}, \ldots, \widetilde{W}_{t+H}$. These sample paths are used to simulate our policy for the remainder of the horizon.

This particular solution to the lookahead policy is unique to this application. However, the idea of adopting a policy in the lookahead model that is easier to compute, while trading off solution quality, is a general strategy that can be applied to any problem. The challenge is identifying that easier-to-compute and good-enough lookahead policy combines art and science.

19.7.2 Lookahead CFAs

From the perspective of choosing lookahead policies, PFAs and CFAs are both parameterized policies that can be approached the same way. There are two major differences when handling CFAs:

- CFAs are computationally more expensive than PFAs, since there is always an imbedded optimization (which may be as simple as a sort, or as complex as an integer program). Imbedded in a DLA, it may be necessary to compute the CFA policy hundreds or thousands of times.
- The search for the best parameter vector θ for a CFA tends to be simpler than that for a PFA, since the CFA incorporates a considerable about of problem structure.

Other than these issues, evaluating CFAs parallels the evaluation of PFAs.

19.7.3 Lookahead VFAs for the lookahead model

VFA-based policies tend to be easier to approximate than PFAs if we do not know the structure of the policy. Another benefit is that it is easier to create time-dependent policies using VFAs than with PFAs. This said, any strategy for solving, even approximately, a dynamic program for each state $\tilde{S}_{tt'}$ to determine $\tilde{x}_{tt'}$ would be hopelessly impractical. Just the same, we have to realize that for problems where a direct lookahead is appropriate, a backward ADP policy (as described in chapter 15), would not be able to capture the complex interactions in the future.

There is another approach that exploits the power of VFA-based policies within a direct lookahead. It consists of two steps:

Step 1: Use backward approximate programming (from chapter 15) once on the lookahead model, which means we are limited to the information in the approximate state $\tilde{S}_{tt'}$ rather than the base state S_t, as we did in chapter 15. From this, we obtain value function approximations $\overline{V}^x_{tt'}(\tilde{S}_{tt'})$ for each time period $t' = t, t+1, \ldots, t+H$ in the lookahead model.

Step 2: Now use these VFAs as the basis for the policy

$$X^{VFA}_{tt'}(\tilde{S}_{tt'}) = \arg\max_{\tilde{x}_{tt'}} \big(C(\tilde{S}_{tt'}, \tilde{x}_{tt'}) + \overline{V}^x_{tt'}(\tilde{S}^x_{tt'})\big),$$

that we then use in step (6a) of the algorithm in figure 19.3. The policy $X^{VFA}_{tt'}(\tilde{S}_{tt'})$ is used to simulate the downstream impact of the decision x_t.

This approach combines the power of a VFA-based policy with the higher level of detail we obtain in a DLA-based policy, since there may be information in the state S_t in the base model that is captured in the lookahead model as latent variables, but not in the lookahead state $\tilde{S}_{tt'}$.

19.7.4 Lookahead DLAs for the lookahead model

So now we return to the idea of using a direct lookahead policy as the lookahead policy in our lookahead model. This is easily going to be the most computationally demanding strategy that we could use for our lookahead policy. We have to first remember that we have to revisit all of the approximation strategies we used to formulate our original lookahead model. We anticipate two in particular will make a DLA policy in the lookahead computationally possible (if not attractive):

- The planning horizon - Let $\tilde{H}$ be the planning horizon within the lookahead DLA model. We would expect $\tilde{H} < H$, but this is clearly a parameter that can be tuned. However, if we aggressively shorten the horizon, we may obtain a practical policy. For example, imagine that we are planning the dispatch of driverless electric vehicles, which means we own the vehicle and need to plan its activities into the future when we make dispatch decisions now. We may decide that we want to plan until the end of the day before making a decision now. However, our policy within the DLA might myopically optimize just one dispatch into the future.

- A deterministic lookahead policy - We might use a deterministic lookahead for the DLA policy within a stochastic model. Using our ride hailing example, we could

sample demands into the future and then solve this optimally, which would give us an optimistic estimate of the value of being in a future state $\tilde{S}_{tt'}$.

Remember that both of these policies have to be solved for each simulated state $\tilde{S}_{tt'}$ that we encounter as we simulate into the future for our DLA policy. Even if these prove to be too expensive, they may serve as benchmarks for other policies.

19.7.5 Discussion

It is not possible to say which of these strategies would be best for a particular problem. Even a single problem class can come in different flavors that require different types of policies. Our goal is to challenge the reader to always consider all four classes of policies, and to use all the tools available. The range of sequential decision problems is vast, and we are trying to make the entire span of the toolbox that the field can offer.

Next, we are going to consider two widely studied (and used) lookahead policies that have attracted considerable attention in the academic literature. The two strategies are:

- Monte Carlo tree search for discrete decisions - This approach has become wildly popular in computer science, largely motivated by its use in games, where direct lookaheads are particularly useful.

- Two-stage stochastic programming for vector decisions - This policy was first developed in the 1950s for a special problem class (make decision, see information, make decision, stop) but has been primarily used as a policy for fully sequential problems.

A third strategy, robust optimization, is one that we introduced earlier in section 19.3 as an example of a lookahead policy with a different objective. This also has attracted considerable attention in the academic literature, although it is unclear how widely it is used in practice. Unfortunately, there is often a big gap between the methods that attract academic attention, and those that are actually used in practice.

19.8 MONTE CARLO TREE SEARCH FOR DISCRETE DECISIONS

For problems where the number of decisions per state is not too large (but where the set of random outcomes may be quite large, even infinite), we may replace the explicit enumeration of the entire tree with a heuristic policy to evaluate what might happen after we reach a state. Imagine that we are trying to evaluate if we should choose decision $\tilde{x}_{tt}$ which takes us to state $\tilde{S}^x_t$ (the post-decision state), after which we choose at random an outcome of $\widetilde{W}_{t,t+1}$, which puts us in the next (pre-decision) state $\tilde{S}_{t,t+1}$. If we repeat this process for each decision $\tilde{x}_{tt}$, and then for each decision $\tilde{x}_{t,t+1}$ out of each of the downstream states $\tilde{S}_{t,t+1}$, the tree would explode in size.

Monte Carlo tree search offers a method that samples the tree in an intelligent way. Given an infinite budget, MCTS will ultimately learn the entire tree, but the hope is that it produces a high quality (perhaps near optimal) initial decision that we can implement in our base model at a reasonable computational cost. As a direct lookahead, MCTS offers a framework where we can draw on many of the other tools we have introduced, but this section will provide a basic introduction to MCTS that is well suited to problems with discrete sets of decisions, and where the set of decisions out of each state is not too large.

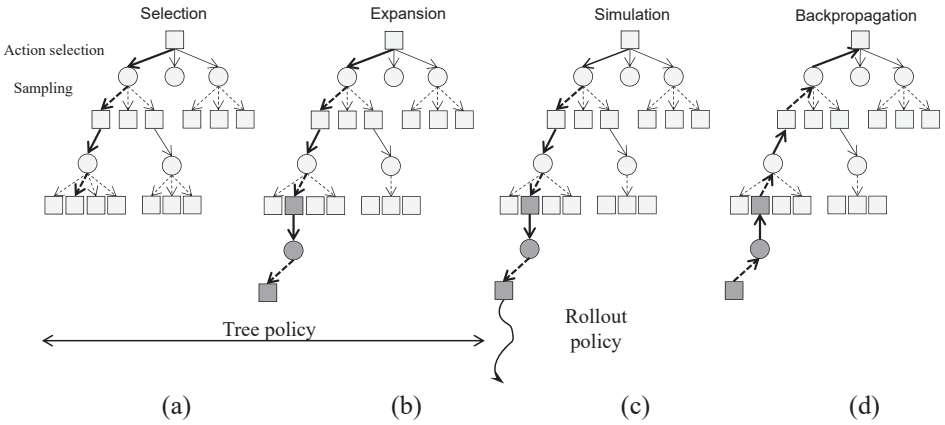

Figure 19.7 Illustration of Monte Carlo tree search, illustrating (left to right): selection, expansion, simulation and backpropagation.

19.8.1 Basic idea

Monte Carlo tree search is a technique that is very popular in computer science (although the roots of the method are in operations research), where it has been primarily used for deterministic problems. MCTS (as it is widely known) proceeds by applying a simple test to each decision out of a node, and then uses this test to choose one decision to explore. This may result in a traversal to a state we have visited before, at which point we simply repeat the process, or we may find ourselves at a new state. We then call a *rollout policy* which is some policy (literally, any of the policies we have discussed up to now) for making decisions that depends on the problem at hand. The rollout policy gives us an estimate of the value of being in this new state. If the state is attractive enough, it is added to the tree.

Each node (state) in the tree is described by four quantities:

1) The pre-decision value function $\widetilde{V}_{tt'}(\tilde{S}_{tt'})$, the post-decision value function $\widetilde{V}^x_{tt'}(\tilde{S}^x_{tt'})$, and the contribution $C(\tilde{S}_{tt'}, \tilde{x}_{tt'})$ from being in state $\tilde{S}_{tt'}$ and taking decision $\tilde{x}_{tt'}$.

2) The visit count, $N(\tilde{S}_{tt'})$, which counts the number of times we have performed rollouts (explained below) from state $\tilde{S}_{tt'}$.

3) The count decision counter, $N(\tilde{S}_{tt'}, \tilde{x}_{tt'})$, which counts the number of times we have taken decision $\tilde{x}_{tt'}$ from state $\tilde{S}_{tt'}$.

4) The set of decisions $\mathcal{X}_s$ from each state s and the random outcomes $\tilde{\Omega}_{t,t'+1}(\tilde{S}^x_{tt'})$ that might happen when in post-decision state $\tilde{S}^x_{tt'}$.

19.8.2 The steps of MCTS

Monte Carlo tree search progresses in four steps which are illustrated in figure 19.7, where the detailed steps of the MCTS are described in a series of procedures. Note that, as before, we let $\tilde{S}_{tt'}$ be the pre-decision state, which is the node that precedes a decision. A

function $MCTS(S_t)$

Step 0. Create root note $\tilde{S}_{tt} = S_t$; set iteration counter $n = 0$.

Step 1. **while** $n < n^{thr}$

Step 1.1 $\tilde{S}_{tt'} \leftarrow TreePolicy(\tilde{S}_{tt})$

Step 1.2 $\widetilde{V}_{tt'}(\tilde{S}_{tt'}) \leftarrow SimPolicy(\tilde{S}_{tt'})$

Step 1.3 $Backup(\tilde{S}_{tt'}, \widetilde{V}_{tt'}(\tilde{S}_{tt'}))$

Step 1.4 $n \leftarrow n + 1$

Step 2. $\tilde{x}_t^* = \arg\max_{\tilde{x}_{tt} \in \tilde{\mathcal{X}}_{tt}(\tilde{S}_{tt})} \tilde{C}(\tilde{S}_{tt}, \tilde{x}_{tt}) + \widetilde{V}_{tt}^x(\tilde{S}_{tt}^x)$

Step 3. **return** x_t^*.

Figure 19.8 Sampled MCTS algorithm.

deterministic function $S^{M,x}(\tilde{S}_{tt'}, \tilde{x}_{tt'})$ takes us to a post-decision state $\tilde{S}_{tt'}^x$, after which a Monte Carlo sample of the exogenous information takes us to the next pre-decision state $\tilde{S}_{t,t'+1}$.

1) Selection - There are two steps in the selection phase. The first (and most difficult) requires choosing a decision, while the second involves taking a Monte Carlo sample of any random information.

   1a) Choosing the decision - The first step from any node (already generated) is to choose a decision (see figure 19.7(a) and the algorithm in figure 19.8). The most popular policy for choosing a decision is to use a type of upper confidence bound (recall we introduced UCB policies in section 7.5) adapted for trees, hence its name, Upper Confidence bounding for Trees (UCT). We could simply choose the decision that appears to be best, but we could get stuck in a solution were we avoid decisions that do not look attractive. Since our estimates are only approximations, we have to recognize that we may not have explored them enough (the classic exploration-exploitation tradeoff). In this setting, the UCT policy is given by

$$X_{tt'}^{UCT}(\tilde{S}_{tt'}|\theta^{UCT}) = \arg\max_{\tilde{x} \in \tilde{\mathcal{X}}_{tt'}} \left(\left(C(\tilde{S}_{tt'}, \tilde{x}) + \widetilde{V}_{tt'}^x(\tilde{S}_{tt'}^x)\right) + \theta^{UCT} \sqrt{\frac{\ln N(\tilde{S}_{tt'})}{N(\tilde{S}_{tt'}, \tilde{x}_{tt'})}} \right)$$

   The parameter θ^{UCT} has to be tuned, just as we would tune any policy. As with UCB policies, the square root term is designed to encourage exploration, by putting a bonus for decisions that have not been explored as often. A nice feature of UCT policies is that they are very easy to compute, which is important in an MCTS setting where we need to quickly evaluate many decisions.

   1b) Sampling the outcome - Here we assume that we can simply take a Monte Carlo sample of any random information (see section 10.4). There are settings where simple Monte Carlo sampling is not very efficient, such as when the random outcome might be a success or failure, where one or the other dominates.

2) Expansion - If the decision we choose above is one we have chosen before, then we progress to the next post-decision state (the solid line connecting the square node to the round node in figure 19.7(b)) at which point we then sample another random

function $TreePolicy(\tilde{S}_{tt})$

Step 0. $t' \leftarrow t$

Step 1. **while** $\tilde{S}_{tt'}$ is non-terminal **do**

Step 2. **if** $|\tilde{\mathcal{A}}_{tt'}(\tilde{S}_{tt'})| < d^{thr}$ **do** (Expanding a decision out of a pre-decision state)

Step 2.1 Choose decision $\tilde{x}^*_{tt'}$ by optimizing on the basis of the contribution of the decision $\tilde{C}(\tilde{S}_{tt'}, \tilde{x}_{tt'})$, then taking a Monte Carlo sample to the next pre-decision state $\tilde{S}_{t,t'+1}$, and then finally using the rollout policy to approximate the value of being in state $\tilde{S}_{t,t'+1}$.

Step 2.2 $\tilde{S}^x_{tt'} = S^M(\tilde{S}_{tt'}, \tilde{x}^*_{tt'})$ (Expansion step)

Step 2.3 $\tilde{\mathcal{X}}_{tt'}(\tilde{S}_{tt'}) \leftarrow \tilde{\mathcal{X}}_{tt'}(\tilde{S}_{tt'}) \bigcup \{\tilde{x}^*_{tt'}\}$

Step 2.4 $\tilde{\mathcal{X}}^u_{tt'}(\tilde{S}_{tt'}) \leftarrow \tilde{\mathcal{X}}^u_{tt'}(\tilde{S}_{tt'}) - \{\tilde{x}^*_{tt'}\}$

else **Step 2.5** $\tilde{x}^*_{tt'} = \arg\max_{\tilde{x}_{tt'} \in \tilde{\mathcal{X}}_{tt'}(\tilde{S}_{tt'})} \left(\left(\tilde{C}(\tilde{S}_{tt'}, \tilde{x}_{tt'}) + \widetilde{V}^x_{tt'}(\tilde{S}^x_{tt'}) \right) + \theta^{UCT} \sqrt{\frac{\ln N(\tilde{S}_{tt'})}{N(\tilde{S}_{tt'}, \tilde{x}_{tt'})}} \right)$

Step 2.6 $\tilde{S}^x_{tt'} = S^M(\tilde{S}_{tt'}, \tilde{x}^*_{tt'})$

end if

Step 3 **if** $|\tilde{\Omega}_{t,t'+1}(\tilde{S}^x_{tt'})| < e^{thr}$ **do** (Expanding an exogenous outcome out of a post-decision state)

Step 3.1 Choose exogenous event $\widetilde{W}_{t,t'+1}$,

Step 3.2 $\tilde{S}_{t,t'+1} = S^{M,x}(\tilde{S}^x_{tt'}, \widetilde{W}_{t,t'+1})$ (Expansion step)

Step 3.3 $\tilde{\Omega}_{t,t'+1}(\tilde{S}^x_{tt'}) \leftarrow \tilde{\Omega}_{t,t'+1}(\tilde{S}^x_{tt'}) \bigcup \{\widetilde{W}_{t,t'+1}\}$

Step 3.4 $\tilde{\Omega}^u_{t,t'+1}(\tilde{S}^x_{tt'}) \leftarrow \tilde{\Omega}^u_{t,t'+1}(\tilde{S}^x_{tt'}) - \{\widetilde{W}_{t,t'+1}\}$

Step 3.5 $t' \leftarrow t' + 1$

return $\tilde{S}_{tt'}$ (stops execution of **while** loop)

else **Step 3.6** Choose exogenous event $\widetilde{W}_{t,t'+1}$,

Step 3.7 $\tilde{S}_{t,t'+1} = S^{M,x}(\tilde{S}^x_{tt'}, \widetilde{W}_{t,t'+1})$

Step 3.8 $t' \leftarrow t' + 1$

end if

end while

Figure 19.9 The tree policy.

outcome which brings us to a new pre-decision state (see the algorithm in figure 19.9). But if we have not chosen this decision before, then we expand our tree by first adding the link associated with the decision to the post-decision state node, followed by a Monte Carlo sample which takes us to the subsequent pre-decision state. At this point we have to deal with the fact that we would not have an estimate of the value of being in this state (which we need for our UCT policy). To overcome this, we call our simulation policy, which is a form of roll-out policy (discussed next).

3) Simulation - The simulation step assumes we have access to some policy which is easy to execute that allows us to obtain a quick and reasonable estimate of the value of being in a state (see figure 19.7(c) and the algorithm in figure 19.10). Of course, this is very problem dependent. Some strategies include:

- A myopic policy, which greedily makes choices. There are problems where myopic policies are reasonable starting estimates (of course they are suboptimal). However, such a greedy policy can be extremely poor (imagine finding

function $SimPolicy(\tilde{S}_{tt'})$

Step 0. Choose a sample path $\tilde{\omega} \in \tilde{\Omega}_{tt'}$

Step 1. **while** $\tilde{S}_{tt'}$ is non-terminal

Step 2.1 Choose $\tilde{x}_{tt'} \leftarrow \tilde{\pi}(\tilde{S}_{tt'})$ where $\tilde{\pi}$ is the rollout policy.

Step 2.2 $\tilde{S}_{t,t'+1} \leftarrow S^M(\tilde{S}_{tt'}, \tilde{x}_{tt'}(\tilde{\omega}))$

Step 2.3 $t' \leftarrow t' + 1$

end while

return $\widetilde{V}_{tt'}(\tilde{S}_{tt'})$ (Value function of $\tilde{S}_{tt'}$)

Figure 19.10 This function simulates the policy.

function $Backup(\tilde{S}_{tt'}, \widetilde{V}_{tt'}(\tilde{S}_{tt'}))$

while $\tilde{S}_{tt'}$ is not null **do**

Step 1.1 $N(\tilde{S}_{tt'}) \leftarrow N(\tilde{S}_{tt'}) + 1$

Step 1.2 $t^* \leftarrow$ t'-1.

Step 1.3 $N(\tilde{S}_{t,t^*-1}, \tilde{x}_{t,t^*-1}) \leftarrow N(\tilde{S}_{t,t^*-1}, \tilde{x}_{t,t^*-1}) + 1$

Step 1.4 $\widetilde{V}^x_{t,t^*-1}(\tilde{S}^x_{t,t^*-1}) \leftarrow \frac{1}{\sum_{\tilde{\omega}_{t,t^*+1} \in \tilde{\Omega}_{t,t^*+1}(\tilde{S}^x_{tt^*})} p(\tilde{\omega}_{t,t^*+1})} \cdot E_g[p(\widetilde{W}_{t,t^*+1})/g(\widetilde{W}_{t,t^*+1})\widetilde{V}_{tt^*}(S^{M,x}(\tilde{S}^x_{tt^*}, \widetilde{W}_{t,t^*+1}))]$

Step 1.5 $\tilde{S}_{tt^*} \leftarrow$ predecessor of $\tilde{S}^x_{tt^*}$

Step 1.6 $\Delta \leftarrow \tilde{C}(\tilde{S}_{tt^*}, \tilde{x}_{tt^*}) + \widetilde{V}^x_{tt^*}(\tilde{S}^x_{tt^*})$

Step 1.7 $\widetilde{V}_{tt^*}(\tilde{S}_{tt^*}) \leftarrow \widetilde{V}_{tt^*}(\tilde{S}_{tt^*}) + \frac{\Delta - \widetilde{V}_{tt^*}(\tilde{S}_{tt^*})}{N(\tilde{S}_{tt^*})}$

Step 1.8 $t' \leftarrow t^*$

end while

Figure 19.11 Backup process which updates the value of each decision node in the tree.

the shortest path through a network by always choosing the shortest link out of a node).

- A parameterized policy with reasonable estimates of the parameters. We might have a rule for selling an asset if its price rises by some percentage. Such a rule will not be ideal, but it will be reasonable.
- A posterior bound. We might sample all future information, and then make the best decision assuming that this future information comes true.

4) Backpropagation - After simulating forward using our rollout policy to obtain an initial estimate of the value of being in our newly generated state, we now backtrack and obtain updated estimates of the value of each of the states on the path to the newly generated state (see figure 19.7(d) and the algorithm in figure 19.11).

Figure 19.12 shows a tree produced by an MCTS algorithm, which illustrates the varying degrees to which MCTS explores the tree. An indication that MCTS is adding value is the presence of narrow sections of the tree which are explored at much greater depth than other portions of the tree. If the tree is fairly balanced, then it means that MCTS is not pruning

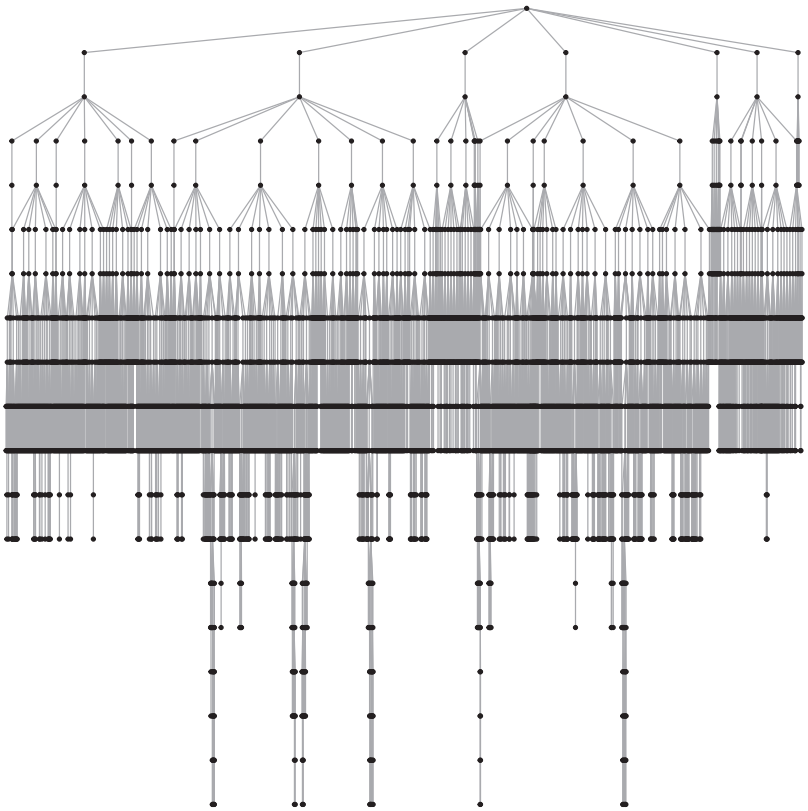

Figure 19.12 Sample of a tree produced by Monte Carlo tree search, illustrating the variable depth produced by an MCTS algorithm.

decisions which means it is basically enumerating the tree. Of course, the real question is how well the resulting tree works as a policy to solve the base model.

19.8.3 Discussion

MCTS is a true mixture - it is a DLA that uses a UCB policy (a form of CFA - see section 7.5), a rollout policy (typically a PFA or CFA), and VFAs. Perhaps the biggest weakness is the rollout policy used to obtain an initial estimate of the value of being at a node. Since we have no guarantees regarding the quality of the rollout policy, the initial estimate will underestimate the true value (on average).

MCTS has been proven to be asymptotically optimal. That is, given an infinite search budget, MCTS will ultimately sample each decision from each state (and all the states will be enumerated) infinitely often. The reason is the UCT policy given by

$$X^{UCT}_{tt'}(\tilde{S}_{tt'}|\theta^{UCT}) = \arg\max_{\tilde{x}\in\tilde{\mathcal{X}}_{tt'}} \left(\left(C(\tilde{S}_{tt'},\tilde{x}) + \widetilde{V}^{x}_{tt'}(\tilde{S}^{x}_{tt'})\right) + \theta^{UCT}\sqrt{\frac{\ln N(\tilde{S}_{tt'})}{N(\tilde{S}_{tt'},\tilde{x}_{tt'})}} \right).$$

The key is the term

$$\sqrt{\frac{\ln N(\tilde{S}_{tt'})}{N(\tilde{S}_{tt'},\tilde{x}_{tt'})}},$$

where $N(\tilde{S}_{tt'})$ is the number of times we visit state $\tilde{S}_{tt'}$ and $N(\tilde{S}_{tt'},\tilde{x}_{tt'})$ is the number of times we visit state $\tilde{S}_{tt'}$ and choose decision $\tilde{x}_{tt'}$. This term goes to infinity as we continue to search the tree. If we do not try decision $\tilde{x}_{tt'}$, then the numerator grows while the denominator holds constant. If we have never tried the decision $\tilde{x}_{tt'}$, then the denominator is zero which forces us to choose among the decisions $\tilde{x}_{tt'}$ that have never been tried until all decisions have been tried at least once (which forces us to explore all the downstream states).

In short, this "uncertainty bonus" term (as it is often known in the bandit community where the logic originated) is the secret that forces us to eventually try everything. This means that even if our rollout policy produces a poor estimate of $\widetilde{V}^{x}_{tt'}(\tilde{S}^{x}_{tt'})$, we will still try every decision $\tilde{x}_{tt'}$, that will eventually force us to visit every state $\tilde{S}_{tt'}$.

Asymptotic convergence proofs need to be taken with a grain of salt (and this includes the proofs in section 5.10). We never run algorithms to the limit, which means what we care about is how well it works within our computation budget. MCTS is no exception. How well it works in practice depends on the computation budget, which depends on choices such as the lookahead policy, as well as the details of how it is coded. For example, there are many opportunities to use parallel computation (we might explore hundreds of decisions all at the same time), which will affect how deep we can explore tree (and how thoroughly) within our budget.

19.8.4 Optimistic Monte Carlo tree search

A variant of MCTS is one that is called *optimistic* MCTS, and it works with one minor modification. Instead of simulating a rollout policy starting at a node $\tilde{S}_{t,t''}$ until the end of the horizon, we generate the sample path of exogenous information $\widetilde{W}_{t,t''}, \widetilde{W}_{t,t''+1}, \ldots, \widetilde{W}_{t,t+H}$. Then, we assume that this entire sample path is known to us, and solve to optimality the deterministic optimization problem that determines the decisions $\tilde{x}_{t,t''}, \tilde{x}_{t,t''+1}, \ldots, \tilde{x}_{t,t+H}$ starting at node $\tilde{S}_{t,t''}$. We then compute the contribution over this path and use it to initialize the value at node $\tilde{S}_{t,t''}$.

By optimizing over the entire sample path, we are making decisions that are allowed to "see" into the future. This produces an optimistic estimate of the value of being at node $\tilde{S}_{t,t''}$ rather than the pessimistic estimate from traditional MCTS. This is a powerful result because it allows us to prove asymptotic optimality *without* requiring that we visit every state, or even test every decision. This is an important feature for problems with a large number of decisions, but it comes at a price. Solving the deterministic optimization problem over the sample path can be quite expensive.

There are two ways to run optimistic MCTS. The least expensive is the logic sketched above where we optimize over the entire sample path, using information that would not otherwise be available. This is known as *information relaxation*. A more advanced algorithm uses penalties that try to enforce the requirement that the decision $\tilde{x}_{tt''}$ be independent of information arriving after t''. Needless to say, this idea is even more expensive.

Figure 19.13 shows the improvement for the following policies as a function of the number of decisions per node:

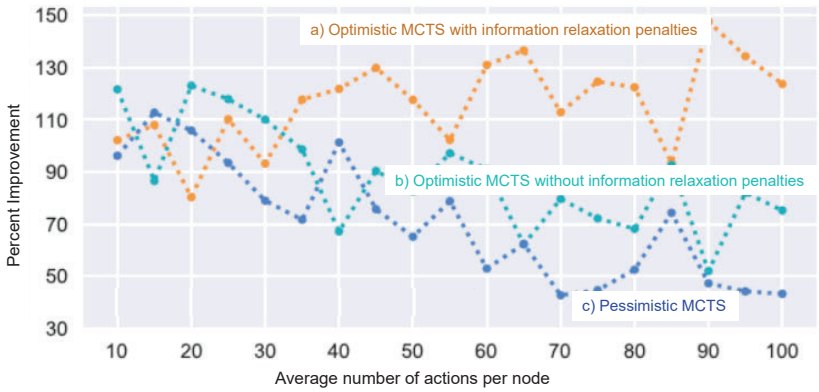

Figure 19.13 Performance of a) optimistic MCTS with information relaxation penalties, b) optimistic MCTS without information relaxation penalties, and c) pessimistic MCTS.

a) Optimistic MCTS where penalties have been added to try to encourage decisions $\tilde{x}_{t,t''}$ that do not reflect future events (that is, $\tilde{w}_{t,t''+1}$ and later).

b) Optimistic MCTS without the information relaxation penalties.

c) The classical "pessimistic" MCTS.

The calculation of the information penalties for (a) involves advanced methods that are beyond the scope of this book, but comparing these three strategies provides some insights into both MCTS as well as two-stage stochastic programming which we describe below.

The results show that optimistic MCTS, with the information relaxation penalties, performs significantly better than pessimistic MCTS as the number of decisions per node exceeds 30. Optimistic MCTS without the information relaxation penalties also outperforms pessimistic MCTS, but the difference is not as dramatic. What this figure does not show is the difference in computation cost. Optimistic MCTS, even without information relaxation penalties, is computationally more expensive than pessimistic MCTS. The computation of the information relaxation penalties imposes an even greater burden. However, this issue has to be addressed in the context of modern computing environments where massive parallelism may be possible.

This is an area that requires more research. It is well known that MCTS is limited to problems where the number of decisions per node is "not too large," but the idea of using optimistic estimates of the value of the future may expand the size of problems (that is, in terms of the number of decisions) that it can be used for.

The idea of making decisions such as $\tilde{x}_{t,t''}$ using information from a (simulated) future is used in other methods, as we will see next with two-stage stochastic optimization. Figure 19.13 is an initial indication of the effect of using information relaxation in a lookahead policy.

19.9 TWO-STAGE STOCHASTIC PROGRAMMING FOR VECTOR DECISIONS*

After deterministic linear programming was invented, researchers quickly realized that there were uncertainties in many applications, leading to an effort (initiated by none other than George Dantzig, who invented the simplex method that started the math programming revolution) to incorporate uncertainty into a linear program.

Not surprisingly, the introduction of uncertainty into mathematical programs opened a Pandora's box of modeling and computational issues that challenges the research community today. For decades the community focused on what became known as the two-stage stochastic programming problem, which consists of "make decision, see information, make one more decision." The two-stage stochastic programming formulation remains a foundational tool for approximating fully sequential problems. Below, we introduce the basic two-stage stochastic programming problem, and then show how it can be used to build a lookahead policy for the fully sequential inventory problem we introduced above. We then show how this can be used as an approximate lookahead model for fully sequential problems.

As a historical note: the field of stochastic programming emerged in the 1950's alongside the development of the field of Markov decision processes by Richard Bellman, with stochastic programming focusing on vector-valued decisions and Markov decision processes working with discrete decisions. These communities evolved in parallel with distinctly different notational systems, modeling frameworks, and relative emphasis on theory vs. computation.

19.9.1 The basic two-stage stochastic program

In section 18.6, we introduced what is known as the *two stage stochastic program* where we make an initial decision x_0 (such as where to locate warehouses), after which we see information $W_1 = W_1(\omega)$ (which might be the demands for product), and then we make a second set of decisions $x_1(\omega)$ which depend on this information (the decisions x_1 are known as the *recourse variables*).

As we did in section 18.6, the two-stage stochastic programming problem is written

$$\max_{x_0} \big(c_0 x_0 + \mathbb{E} V_1(x_0, W_1)\big), \tag{19.42}$$

subject to the constraints,

$$A_0 x_0 = b_0, \tag{19.43}$$

$$x_0 \geq 0, \tag{19.44}$$

The initial decisions x_0 (which determines the inventories in the warehouses) then impacts the decisions that can be made after the information (the demand) becomes known, producing the second stage problem

$$V_1(x_0, \omega) = \max_{x_1(\omega)} c_1(\omega) x_1(\omega), \tag{19.45}$$

subject to, for all $\omega \in \Omega$,

$$A_1 x_1(\omega) \leq B_1 x_0, \tag{19.46}$$

$$B_1 x_1(\omega) \leq D_1(\omega), \tag{19.47}$$

$$x_1(\omega) \geq 0. \tag{19.48}$$

Note that we write the value function for the second state, $V_1(x_0, W_1)$ in terms of x_0, although x_0 is not a proper state variable. For example, we could write

$$R_1 = B_1 x_0,$$

and then write the first stage objective as

$$\max_{x_0} \big(c_0 x_0 + \mathbb{E} V_1(R_1, W_1)\big).$$

In fact, for many applications R_1 is lower dimensional (and possibly much lower dimensional) than x_0. For example, R_1 might be a vector of inventories, where R_{1i} is the amount of inventory at location i, while x_0 is a high dimensional vector with elements x_{0ij}. Our version above represents standard convention in this community.

We cannot actually compute this model if Ω represents all the potential outcomes, so we have to use the idea of a sampled model that we first introduced in section 4.3. We note that even when Ω is carefully designed and "not too big," the two-stage problem can still be hard to solve (if the decision vectors are large enough).

There are several computational strategies that have been used to solve problem (19.42) - (19.48):

The "deterministic equivalent" method - The problem (19.42) - (19.48), when formulated using a sampled set of observations Ω, is basically a single (potentially large) deterministic linear program, leading some to refer to this problem as the "deterministic equivalent." Modern solvers can handle problems with hundreds of thousands of variables with minimal training (specialists have worked on problems with millions of variables).

Relaxation - If we replace x_0 with $x_0(\omega)$, this means that we are allowing x_0 to see the future. We can fix this with a *nonanticipativity constraint* that looks like

$$x_0(\omega) = x_0, \text{ for all } \omega \in \hat{\Omega}. \tag{19.49}$$

This formulation allows us to design algorithms that relax this constraint, allowing us to solve $|\Omega|$ independent problems with logic that penalizes deviations of (19.49). This methodology has become known under the name of *progressive hedging* which has made it possible to approach two-stage stochastic programs that would otherwise be too large.

Benders decomposition - In section 18.6 we introduced the idea of using Benders decomposition, where the function $\mathbb{E} V_1(x_0, W_1)$ is replaced with a series of cuts. This is really a form of approximate dynamic programming, which we showed in chapter 18.

We are now going to transition to using this two-stage model as a policy for fully sequential problems.

19.9.2 Two-stage approximation of a sequential problem

While there are true two-stage stochastic programming problems, there is a vast range of fully sequential problems of the types that we have been pursuing in this book that involve vector-valued decisions in the presence of uncertainty. A widely used strategy is to solve

these problems by approximating them as two-stage problems, where there is a decision to be made now at time t, represented by x_t, after which we then pretend that we observe all the future information over the rest of our horizon $(t+1, t+H)$. After this information is revealed, we then make all remaining decisions $\tilde{x}_{tt'}$ for $t' = t+1, \ldots, t+H$, which represents (along with the revealed information) the second stage of our decision problem.

We are going to illustrate these ideas using the energy storage problem we first addressed in section 13.3.3, where we are pulling energy from the grid (at a random price), a wind farm (with random supply), to meet a time-dependent load, with a storage device to smooth out the different sources of variability. It may be helpful to flip back to this section to review the model, but we will repeat the definitions of a few variables. Our state variables were

D_t = Demand ("load") for power during hour t.
E_t = Energy generated from renewables (wind/solar) during hour t.
R_t = Amount of energy stored in the battery at time t.
u_t = Limit on how much generation can be transmitted at time t (this is known in advance).
p_t = Price to be paid for energy drawn from the grid at time t.

The decision variables were given by

x_t^{ED} = Flow of energy from wind to demand,
x_t^{EB} = flow of energy from wind to battery,
x_t^{GD} = flow of energy from grid to demand,
x_t^{GB} = flow of energy from grid to battery,
x_t^{BD} = flow of energy from battery to demand.

Let x_t be the vector of all five flows, and let $\tilde{x}_{tt}, \ldots, \tilde{x}_{t,t+H}$ be the set of vectors over our planning horizon. As we did in section 13.3.3, we want to plan over a horizon where we no longer use point forecasts of the demands $D_{t'}$ and energy from wind or solar, $E_{t'}$. Instead, we want to explicitly model the uncertainty in $D_{t'}$, $E_{t'}$ and $p_{t'}$ for $t' > t$.

First, we are going to create a set of M sample paths that we call $\tilde{\Omega}_t$ (indexed by t because it is generated at time t), where each $\tilde{\omega}_t \in \tilde{\Omega}_t$ represents a full sequence of the random variables $W_{t+1}, W_{t+2}, \ldots, W_{t+H}$. Given $\tilde{\omega}_t \in \tilde{\Omega}_t$ (that is, given the rest of the future), we index all future decisions with the same outcome $\tilde{\omega}_t$, giving us the vector $\tilde{x}_{tt'}(\tilde{\omega}_t)$, for $t' = t+1, t+2, \ldots, t+H$. The outcomes $\tilde{\omega}_t$ are known as *scenarios* in the language of stochastic programming.

Next we are going to make decisions $\tilde{x}_{t,t+1}(\tilde{\omega}_t), \tilde{x}_{t,t+2}(\tilde{\omega}_t), \ldots, \tilde{x}_{t,t+H}(\tilde{\omega}_t)$ for each sample path $\tilde{\omega}_t$. Since $\tilde{\omega}_t$ indexes the entire sample path, that means each $\tilde{x}_{tt'}(\tilde{\omega}_t)$ is allowed to "see" the entire future. However, we only allow this for decisions $\tilde{x}_{tt'}$ in the future (that is, $t' > t$). The current decision that we would implement, $\tilde{x}_{tt}$, is not indexed by $\tilde{\omega}_t$.

We now have the following sets of variables:

- Variables indexed by (tt) (that is, known or determined now), which include:
  - Parameters or quantities, including costs and constraints.
  - Decisions $\tilde{x}_{tt}$ which will be implemented.

- Variables indexed by (tt') for $t' > t$, which include:
  - Parameters or quantities in the future which are not known now.
  - Decisions $\tilde{x}_{tt'}$ that we plan to help us make $\tilde{x}_{tt}$, but which will not be implemented.

There are two ways to model $\tilde{x}_{tt}$. The first is to treat it as a single vector that must be determined now. The second is to allow it to also be indexed by $\tilde{\omega}_t$, creating a family of variables $\tilde{x}_{tt}(\tilde{\omega}_t)$, each of which is allowed to see the entire sample path (which means it is allowed to see into the future). This raises the question: which one to implement? We need just a single decision to be implemented, which we do by imposing the constraint

$$\tilde{x}_{tt}(\tilde{\omega}_t) = x_t. \tag{19.50}$$

Equation (19.50) is known in the stochastic programming literature as a *nonanticipativity constraint*, which we first saw in chapter 2 (see equation (2.25)). The first question that a reader should ask is: why would we even use the notation $x_{tt}(\omega)$? The reason is computational. Imagine that we write the time t decision as $x_{tt}(\omega)$ and temporarily ignore equation (19.50). In this case, the problem would decompose into a series of problems, one for each $\tilde{\omega}_t \in \tilde{\Omega}_t$. These are much smaller problems than a single problem where we have to deal with all the different scenarios at the same time, but it means that we can get a different answer $\tilde{x}_{tt}(\tilde{\omega}_t)$ at time t, which is a problem. However, there are algorithmic strategies that take advantage of this problem structure.

We can model the first period decision as $\tilde{x}_{tt}(\tilde{\omega}_t)$ and impose a nonanticipativity constraint (19.50), or we can simply use x_t and just let all the future decisions $\tilde{x}_{tt'}(\tilde{\omega}_t)$ for $t' > t$ depend on the scenario. If we use the latter formulation (which is simpler to write, but may be harder to solve), we can write our policy at time t as

$$X_t^{SP}(S_t|\theta) = \arg\min_{x_t} \Bigg(p_t\big(x_t^{GB} + x_t^{GD}\big) + \min_{(\tilde{x}_{tt'}(\tilde{\omega}_t))_{t'=t+1}^{t+H},\, \tilde{\omega}_t \in \tilde{\Omega}_t} \sum_{\tilde{\omega}_t \in \tilde{\Omega}_t} P(\tilde{\omega}_t) \sum_{t'=t+1}^{t+H} \big(\tilde{p}_{tt'}(\tilde{\omega}_t)\big(\tilde{x}_{tt'}^{GB}(\tilde{\omega}_t) + \tilde{x}_{tt'}^{GD}(\tilde{\omega}_t)\big) - \theta^{pen}\tilde{x}_{tt'}^{slack}(\tilde{\omega}_t)\big) \Bigg) \tag{19.51}$$

subject to the constraints

$$\tilde{R}_{t'+1}(\tilde{\omega}_t) - \big(\tilde{x}_{tt'}^{GB}(\tilde{\omega}_t) + \tilde{x}_{tt'}^{EB}(\tilde{\omega}_t) - \tilde{x}_{tt'}^{BD}(\tilde{\omega}_t)\big) = \tilde{R}_{t'}(\tilde{\omega}_t), \tag{19.52}$$

$$\tilde{x}_{tt'}^{ED}(\tilde{\omega}_t) + \tilde{x}_{tt'}^{EB}(\tilde{\omega}_t) \leq \tilde{h}_{t'}(\tilde{\omega}_t), \tag{19.53}$$

$$\tilde{x}_{tt'}^{BD}(\tilde{\omega}_t) + \tilde{x}_{tt'}^{GD}(\tilde{\omega}_t) + \tilde{x}_{tt'}^{ED}(\tilde{\omega}_t) + \tilde{x}_{tt'}^{slack}(\tilde{\omega}_t) = \tilde{D}_{t'}(\tilde{\omega}_t), \tag{19.54}$$

$$\tilde{x}_{tt'}^{GB}(\tilde{\omega}_t), \tilde{x}_{tt'}^{EB}(\tilde{\omega}_t), \tilde{x}_{tt'}^{BD}(\tilde{\omega}_t), \tilde{x}_{tt'}^{ED}(\tilde{\omega}_t), \tilde{x}_{tt'}^{slack}(\tilde{\omega}_t) \geq 0. \tag{19.55}$$

Equation (19.52) is the flow conservation constraint for our battery, while equation (19.53) limits the power available from the wind or solar farm. Equation (19.54) limits how much we can deliver to the customer, where $\tilde{x}_{tt'}^{slack}$ is the slack variable that captures how much demand is not satisfied, which carries a penalty θ^{pen} in the objective function.

This formulation allows us to make a decision at time t while modeling the stochastic variability in future time periods. In the stochastic programming community, the outcomes $\tilde{\omega}_t$ are often referred to as *scenarios*. If we model 20 scenarios, then our optimization problem becomes roughly 20 times larger, so the introduction of uncertainty in the future

comes at a significant computational cost. However, this formulation allows us to capture the variability in future outcomes, which can be a significant advantage over using simple forecasts. Furthermore, while the problem is certainly much larger, we can approach it using one of the three algorithmic strategies described in section 19.9.1.

Although the two-stage approximation is dramatically smaller than a full multistage model, even two-stage approximations can be quite challenging. The problem is that there are many applications where even a deterministic lookahead model can be hard to solve. Fortunately, modern solvers for linear, integer and convex optimization problems have improved dramatically over the years, along with the speed of computers and the ability to support parallelization for optimization solvers.

19.9.3 Discussion

The stochastic programming community almost universally views the two-stage lookahead policy as "the problem" that they are solving. Keep in mind that there are many applications (often with integer variables) where the two-stage stochastic program is quite challenging to solve. Research teams have sometimes spent years developing these models. Overlooked is the reality that the problem defined by (19.51)-(19.55) is actually a policy to solve our base model (such as that given in equation (19.1)). As with all approximate lookahead models (and there are a number of approximations in this model), an optimal solution of an approximate lookahead model is never an optimal policy.

Most users of two-stage stochastic programming will focus on the approximation of using a sample of scenarios. Often overlooked, surprisingly, are the errors from the two-stage approximation, where decisions $\tilde{x}_{t,t+1}(\tilde{\omega}_t), \tilde{x}_{t,t+2}(\tilde{\omega}_t), \ldots, \tilde{x}_{t,t+H}(\tilde{\omega}_t)$ are allowed to see the entire sequence of exogenous random variables of prices, demands, and energy from renewables. The effect of this approximation is, of course, highly problem dependent, but this is where we refer back to the simulations reported in figure 19.13 for MCTS which indicates that the two-stage approximation may in fact produce a significant reduction in the quality of the policy when simulated in the base model.

19.10 OBSERVATIONS ON DLA POLICIES

A number of comments are in order regarding the use of lookahead policies:

- It is common when using lookahead policies to form a model, and then assume that this is the model we have to solve. The real problem is the base model, while the lookahead policy is just one way of creating a policy for the base model.

- It is sometimes hard to know if a model is a lookahead model or a base model. There are many instances of stochastic dynamic programs which are base models, but these might also be lookahead models. We provide some guidance below.

- Optimal solutions of stochastic lookahead models can be quite difficult to find, but an optimal solution of an approximate lookahead model is not an optimal policy for the base model, which is the real model being solved.

- While simulating a policy (any policy) is typically the best way to tune and compare policies, this is critical when using policy search (policy function approximations, or cost function approximations). By contrast, it is less important when using lookahead

models. Given the complexity of building simulators, along with the approximations inherent in any simulation, there are many situations where lookahead policies are just tested in the field. When this process is used, it is important to recognize that this is happening, and to collect performance metrics to evaluate the performance of the policy, and to assess changes made in the policy.

As of this writing, the vocabulary of "base models" and "lookahead models" has not entered the language of modeling in stochastic optimization. As a result, it can be difficult to identify whether a stochastic optimization model is a base model or lookahead model. Some guidelines include:

- How is the model being used? If the primary output of a model is the decision we make in the first time period, this is almost always a lookahead model. On the other hand, it is possible the model is being used to answer strategic questions, where we are using optimization to simulate good decisions. In this case, the model is a base model. Note that the same model can serve as either a lookahead model or a base model.
- If we are using a deterministic lookahead model, or a stochastic lookahead using scenario trees, to make decisions in a setting that is changing dynamically, then this is a lookahead model. The future decisions in the lookahead model (that is, $\tilde{x}_{tt'}$ for $t' > t$) are not going to be implementable in a stochastic setting.

Perhaps the most important take-away of this chapter is the need to clearly distinguish the lookahead model from the base model, and to remember that the problem we are trying to solve is the base model. Then, the challenge is to design a lookahead model that strikes a balance between realism and computational tractability, especially in the design of the lookahead policy. Our goal has been to provide a menu of choices along both dimensions, with the hope that some combination will work for a specific application.

19.11 BIBLIOGRAPHIC NOTES

Section 19.1 - Lookahead models fall in the broad class of policies known in the controls community as "model predictive control" (or MPC), although MPC is often equated with deterministic lookaheads. For a deterministic base model, a deterministic lookahead, which looks all the way to the end of the horizon, is an optimal policy (see Morari et al. (2002) and Camacho & Bordons (2003)). For stochastic problems, a full stochastic lookahead is equivalent to Bellman's optimality principle (see Puterman (2005)).

Section 19.2 - Our method of developing the approximate lookahead, including the identification of the different types of approximations and the notation, was first presented in Powell (2014).

Section 19.3 - The idea of using modified objectives in a lookahead policy, while still using expectations to evaluate the policy, has been used in an ad-hoc manner. For example, Ben-Tal et al. (2005) uses a robust optimization objective to solve an inventory problem, but then states (p. 262):

> "To evaluate the actual outcomes that might result from employing this model, we ran hundreds of simulations with different data sets and compared the mean performance vis-à-vis the mean PH [planning horizon] outcome."

In other words, they are simulating the robust policy hundreds of time and taking an average (that is, approximating an expectation). We believe that simulating policies that are computed using various risk metrics, is probably quite common, but it is done (as was the case in Ben-Tal et al. (2005)) without an explicit model of the policy.

Section 19.3.1 - The subject of risk in sequential decision problems under uncertainty is exceptionally risk, with intellectually deep issues and mathematics. At the heart of the issue is the (typical) lack of additivity of risk across time periods, something we have taken for granted when working with expectations. It was the work of Andrzej Ruszczynski on dynamic risk measures (starting with Ruszczyński (2010)) that largely launched the study of risk measures in dynamic programs. For an introduction to this line of research, we recommend starting with his tutorial Ruszczyński (2014). Philpott & De Matos (2012) provides a very nice summary of nested risk measures in a direct lookahead policy for a water reservoir planning model for New Zealand which is solved using SDDP (from chapter 18). The paper then illustrates the subsequent simulation of the policy, where both the average performance (which determines the cost of operation) and the risk of power outages are considered. Shapiro et al. (2013) illustrates these issues for a hydroelectric planning problem for Brazil. Maceira et al. (2014) addresses risk in setting spot prices for the Brazilian system. Collado et al. (2017) illustrate risk modeling for a risk-averse stochastic path detection problem.

Section 19.6 - Deterministic lookaheads are a widely used heuristic for making sequential decisions in dynamic environments. This is known as "model predictive control" in the optimal control literature, which is the only community to have seriously studied the properties of deterministic lookahead policies, although this is all in the setting of deterministic problems (see Morari et al. (2002) and Camacho & Bordons (2003)). Secomandi (2008) studies the effect of reoptimization on rolling horizon procedures as they adapt to new information.

Section 19.7 - The idea of using any of the four classes of policies (or a hybrid) as the lookahead policy is new, although MCTS is an inherently hybrid approach.

Section 19.8 - The foundation for the ideas behind MCTS are based on Chang et al. (2005). The first use of "Monte Carlo tree search" appeared in Coulom (2006). A review of Monte Carlo tree search is given in Browne et al. (2012), although this is primarily for deterministic problems. Other recent reviews include Auger et al. (2013) and Munos (2014). For a recent review of MCTS and its application in games such as computer Go, see Fu (2017). A nice tutorial on MCTS is given in Fu (2018).

Central to the success of MCTS is having an effective rollout policy to get an initial approximation of the value of being in a leaf node. Rollout policies were originally introduced and analyzed in Bertsekas et al. (1997) and Bertsekas & Castanon (1999).

Section 19.8.4 - Optimistic MCTS is based on the idea of using a deterministic sample in the estimation of the value of a node, and then optimizing using the complete future, which is a form of information relaxation, producing an optimistic estimate of the value of being at a node. Jiang et al. (2020) presents an asymptotic proof of convergence of optimistic MCTS.

Section 19.9 - Two-stage resource allocation problems are widely used to illustrate general two-stage stochastic programming, first introduced by Dantzig (1955), which

launched the field of stochastic programming (see Birge & Louveaux (2011), Kall & Wallace (2009), Shapiro et al. (2014)). We are not aware of any analysis of the performance of two-stage stochastic programming models when used as policies in sequential resource allocation problems.

EXERCISES

Review questions

19.1 Describe at least three examples of problems where a) it seems apparently that you need to anticipate the downstream impact of a decision now and b) you do not think you can reasonably approximate this using a value function approximation. Feel free to use settings from your own personal experiences.

19.2 The variables in the lookahead model, $\tilde{S}_{tt'}$, $\tilde{x}_{tt'}$, and $\widetilde{W}_{tt'}$, are all indexed by two time indices (t and t'). Explain the purpose of each time index.

19.3 List the six classes of approximation strategies that can be applied to develop an approximate lookahead model. Briefly explain each.

19.4 List three examples (not given in section 19.3.1) of risk. These may be drawn from any setting.

19.5 Since you have already decided to use a direct lookahead policy (at least, that is why you are reading this chapter). Why aren't you use a DLA for your choice of "policy within a policy"? What are the major concerns when designing the "policy within a policy"?

Problem solving questions

19.6 If we are using a PFA (parameterized by θ) as our lookahead policy (see equation (19.40)), we need to optimize θ. What is the objective function that we should, in theory, use to optimize θ? What does your optimal θ depend on?

19.7 We are trying to find the best path through a dynamic transportation network, so that we travel from origin r to destination s, and arrive at the destination node s by a specific time t^{target}. We want to

a) Begin by modeling the deterministic shortest path problem as a lookahead model, but modified so that you capture how late you are relative to t^{target} when you arrive at the destination node s. What is the state variable that you need for this problem?

b) Given the Bellman equation for this deterministic approximation.

c) Assume that you are going to solve this problem using updated estimates of travel costs as you arrive to each node. Model the state variable for this sequential decision problem.

d) Give the expression for simulating the performance of this policy for a sample path of updated costs.

19.8 In our presentation of the dynamic shortest path problem (see section 13.2.3), we noted that the state variable (assuming the traveler takes one time period to traverse each

link in the network) was

$$S_t = (R_t, \tilde{c}_t).$$

where R_t captures the node where the traveler is currently located (which means they have to make a decision), and $\tilde{c}_t = (\tilde{c}_{t,k\ell})_{k,\ell\in\mathcal{N}}$ is the vector of the current estimates of the travel time on each link (k, ℓ), which is updated according to

$$\tilde{c}_{t+1,k\ell} = \tilde{c}_{t,k\ell} + \delta\tilde{c}_{t+1,k\ell}.$$

The standard way of solving this problem is to create a lookahead policy that requires solving a deterministic shortest path problem.

a) What approximations are being made when we replace the original problem with the deterministic shortest path problem. Be specific (it is not enough to just say "it is deterministic.")

b) What is the state variable of the deterministic problem?

c) One approach for incorporating uncertainty is to replace the point estimate (as of time t) of $\tilde{c}_{t,k\ell}$ with the $\tilde{c}^{\pi}_{t,i,j}(\theta)$ which is defined by

$$\tilde{c}^{\pi}_{t,i,j}(\theta) \quad = \quad \text{the } \theta\text{-percentile of the travel time for link } (i,j) \text{ given our estimate at time } t.$$

Rewrite the policy in (19.5) using the parameterized lookahead policy and describe how we are supposed to go about optimizing θ. Explain why the optimal value of θ is a function, and describe what it is a function of.

d) Rewrite the policy in equation (19.5) if we fix define θ to be a scalar. Give the objective function for optimizing θ.

19.9 You are tasked with purchasing futures contracts for natural gas. Let $x_{tt'}$ be the quantity of natural gas that you wish to purchasing now for delivery at time t', at a price $p_{tt'}$. The prices are seasonal, but with a fair amount of noise. If you purchase more than you need, you have the ability to store the excess. Let u be the capacity of the storage reservoir for natural gas. Assume you initially have R_0.

Create a lookahead policy for helping to make the set of decisions $x_t = (x_{tt'})_{t'\geq t}$ that need to be implemented now. Remember that you will have to model decisions $\tilde{x}_{t,t',t''}$ in the lookahead model, which represent contracts that we *plan* now, but where we do not actually make the decision to place the purchase until time t' for deliveries at time t''. Let $\tilde{p}_{t,t',t''}$ be the forecast of the forward price for natural gas that might be in place at time t' for delivery at time t''. Note that these forecast evolve continuously according to the MMFE model (see, for example, section 9.4). Do not worry about computing the lookahead policy.

19.10 Using your model from exercise 19.9, create a deterministic version of your lookahead model.

a) In view of the seasonal nature of the demand for natural gas (highest in the winter for heating), and the capacity constraint on storage, common on the strengths of a deterministic lookahead mode.

b) Now consider the potential price variations (which can be substantial) but which would never be forecasted (price spikes occur for very transient reasons), common

on the behaviors that you would like in a policy, but which a deterministic lookahead model might miss.

19.11 Using your model from exercise 19.9, and your deterministic lookahead from exercise 19.10, design a parameterized lookahead that will produce a policy that will work better given the uncertainty of price spikes. Your parameters should take on values such as 0 (if additive) or 1 (if multiplicative) in the event that there are no price variations. Show how your parameterized policy can be tuned so that it will be better positioned to handle price spikes (assume that these happen at random, and cannot be forecasted).

Sequential decision analytics and modeling

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

19.12 Read sections 6.1-6.4 on the dynamic shortest path problem. The method describes using dynamic programming to solve a deterministic approximation of the network, and uses this as a deterministic lookahead policy. The algorithm has been implemented in Python, which can be downloaded from `http://tinyurl.com/sdagithub` using the module "StochasticShortestPath_Dynamic."

a) Use the deterministic lookahead model as a policy (which is already coded) and simulate the policy on the network that is provided. Report the performance of the policy in terms of the time required to complete the path.

b) Now, modify the costs on the links so that you are using the θ-percentile rather than the mean. Use the assumed mean and standard deviation for the random costs $\hat{c}_{ij}$ to create a θ-percentile cost $\bar{c}(\theta)_{ij}$. This is a new deterministic shortest path problem. Modify the code to simulate the performance of this policy for $\theta = 0.5, 0.7, 0.9$. Simulate each policy over 20 samples, and report the mean and standard deviation for each value of θ.

Diary problem

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

19.13 Choose a decision in your diary problem that might reasonably be made using a direct lookahead policy (recall that *any* decision can, in principle, be made using a direct lookahead policy). Now answer the following:

a) Describe the decision and how the decision impacts the future.

b) Assess the pros and cons of using a deterministic lookahead model for a policy. Do you feel that this would be a good approximation?

c) Suggest an approximate lookahead model, and describe at least one candidate for the "policy within a policy."

Bibliography

Auger, D., Couëtoux, A. & Teytaud, O. (2013), Continuous upper confidence trees with polynomial exploration - Consistency, *in* 'Joint European Conference on Machine Learning and Knowledge Discovery in Databases.', Springer, pp. 194–209.

Ben-Tal, A., Golany, B., Nemirovski, A. & Vial, J.-P. (2005), 'Retailer-Supplier Flexible Commitments Contracts: A Robust Optimization Approach', *Manufacturing & Service Operations Management* **7**(3), 248–271.

Bertsekas, D. P. & Castanon, D. A. (1999), 'Rollout Algorithms for Stochastic Scheduling Problems', *J. Heuristics* **5**, 89–108.

Bertsekas, D. P., Tsitsiklis, J. N. & Wu, C. (1997), 'Rollout Algorithms for Combinatorial Optimization', *Journal of Heuristics* **3**(3), 245–262.

Birge, J. R. & Louveaux, F. (2011), *Introduction to Stochastic Programming*, 2nd edn, Springer, New York.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S. & Colton, S. (2012), 'A Survey of Monte Carlo Tree Search Methods', *IEEE Trans. on Computational Intelligence and AI in Games* **4**(1), 1–49.

Camacho, E. & Bordons, C. (2003), *Model Predictive Control*, Springer, London.

Chang, H. S., Lee, H.-g., Fu, M. C. & Marcus, S. I. (2005), 'Evolutionary policy iteration for solving Markov Decision processes', *IEEE Trans. Automat. Control* **50**(11), 1804–1808.

Collado, R., Meisel, S. & Priekule, L. (2017), 'Risk-averse stochastic path detection', *European Journal of Operational Research* **260**(1), 195–211.

Coulom, R. (2006), Efficient selectivity and backup operators in Monte-Carlo tree search, *in* 'Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)', Springer-Verlag, New York, pp. 72–83.

Dantzig, G. B. (1955), 'Linear programming with uncertainty', *Management Science* **1**, 197–206.

Fu, M. C. (2017), Markov Decision Processes, AlphaGo, and Monte Carlo Tree Search: Back to the Future, *in* 'TutORials in Operations Research', pp. 68–88.

Fu, M. C. (2018), Monte Carlo tree search: A tutorial, *in* 'Winter Simulation Conference', pp. 222–236.

Jiang, D. R., Al-kanj, L. & Powell, W. B. (2020), 'Optimistic Monte Carlo Tree Search with Sampled Information Relaxation Dual Bounds Optimistic Monte Carlo Tree Search with Sampled Information Relaxation Dual Bounds', (September), 0–20.

Kall, P. & Wallace, S. W. (2009), *Stochastic Programming*, Vol. 10, John Wiley & Sons, Hoboken, NJ.

Maceira, M. E., Marzano, L. G., Penna, D. D., Diniz, A. L. & Justino, T. C. (2014), 'Application of CVaR risk aversion approach in the expansion and operation planning and for setting the spot price in the Brazilian hydrothermal interconnected system', *Proceedings - 2014 Power Systems Computation Conference, PSCC 2014*.

Morari, M., Lee, J. H. & Garc, C. E. (2002), *Model Predictive Control*, Springer-Verlag, New York.

Munos, R. (2014), *From Bandits to Monte-Carlo Tree Search: The Optimistic Principle Applied to Optimization and Planning*, Vol. 7.

Philpott, A. B. & De Matos, V. L. (2012), 'Dynamic sampling algorithms for multi-stage stochastic programs with risk aversion', *European Journal of Operational Research* **218**(2), 470–483.

Powell, W. B. (2014), 'Clearing the Jungle of Stochastic Optimization', *Informs TutORials in Operations Research 2014*.

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

Ruszczyński, A. (2010), 'Risk-averse dynamic programming for Markov decision processes', *Mathematical Programming* **125**, 235–261.

Ruszczyński, A. (2014), Advances in Risk-Averse Optimization, *in* 'Informs Tutorials in Operations Research', Informs, Baltimore, MD, chapter 9, pp. 168–190.

Secomandi, N. (2008), 'An Analysis of the Control-Algorithm Re-solving Issue in Inventory and Revenue Management', *Manufacturing & Service Operations Management* **10**(3), 468–483.

Shapiro, A., Dentcheva, D. & Ruszczyński, A. (2014), *Lectures on Stochastic Programming: Modeling and theory*, 2 edn, SIAM, Philadelphia.

Shapiro, A., Tekaya, W., Da Costa, J. P. & Soares, M. P. (2013), 'Risk neutral and risk averse Stochastic Dual Dynamic Programming method', *European Journal of Operational Research* **224**(2), 375–391.