Clearing the Jungle of Stochastic Optimization

Warren B. Powell

Department of Operations Research and Financial Engineering
Princeton University

Prepared for Informs TutORials, 2014.

DRAFT

April 7, 2014

Abstract

While deterministic optimization enjoys an almost universally accepted canonical form, stochastic optimization is a jungle of competing notational systems and algorithmic strategies. This is especially problematic in the context of sequential (multistage) stochastic optimization problems, which is the focus of our presentation. In this article, we place a variety of competing strategies into a common framework which makes it easier to see the close relationship between communities such as stochastic programming, (approximate) dynamic programming, simulation, and stochastic search. What have previously been viewed as competing approaches (e.g. simulation vs. optimization, stochastic programming vs. dynamic programming) can be reduced to four fundamental classes of policies that are evaluated in a simulation-based setting. The result is a single coherent framework that encompasses all of these methods, which can often be combined to create powerful hybrid policies to address complex problems.

Contents

1 Introduction

Arguably one of the most familiar pieces of mathematics in operations research (and certainly in optimization) is the linear program, almost always written in its canonical form

$$\min_x c^T x \tag{1}$$

subject to

$$\begin{aligned} Ax &= b, & (2)\\ x &\geq 0. & (3) \end{aligned}$$

Often we are solving problems over time (the focus of this paper). If our problem is deterministic, we would rewrite (1)-(3) as

$$\min_{x_0,\ldots,x_T} \sum_{t=0}^{T} c_t^T x_t \tag{4}$$

subject to

$$\begin{aligned} A_0 x_0 &= b_0, & (5)\\ A_t x_t - B_{t-1} x_{t-1} &= b_t, \quad t = 1,\ldots,T, & (6)\\ x_t &\geq 0, \quad t = 1,\ldots,T. & (7) \end{aligned}$$

First introduced by Kantorovich (1939) and then by Koopmans (1947), it was made famous by Dantzig with the introduction of the simplex method (a nice review of the history is given in Dorfman (1984)) which transformed equations (1)-(3) (or (4)-(7)) from a mathematical statement to a powerful tool for solving a wide range of problems. While Dantzig is most often credited with inventing the simplex method, the canonical form of a linear program is widely used (especially for integer programs) even when the simplex method is not used. The language of equations (1)-(3) is spoken around the world, by academics and industrial practitioners alike.

Now consider what happens when we introduce a random variable, especially for multiperiod problems which require solving a sequence of decision problems interspersed with the revelation of new information. Suddenly, the academic community starts speaking a dozen languages which can quickly become arcane. It is hard to read more than a handful of papers without running into terms such as admissible policies, $\mathcal{F}_t$-measurability, sigma-algebras, and filtrations. Not familiar with these terms? Pity you. You may not have what it takes to work in the rarefied world of stochastic optimization.

But wait... we all seem to make decisions over time, every day, in the presence of all sorts of uncertainties. Think about the last time you took cash out of an ATM machine, chose a path through a set of city streets, made an investment, or tried a new restaurant. Companies invest in new projects, design products, and set pricing strategies. The health community runs tests and prescribes medications, while scientists around the world design and run experiments to make new discoveries. All of these are stochastic optimization problems. If this is something we all do, all the time, why is stochastic optimization so arcane?

After floundering around the fields of stochastic optimization for 30 years, my conclusion is that we have evolved solution strategies around specific problem classes, with vocabulary and notation that often disguise common themes. These solution strategies evolve within communities with names such as dynamic programming, stochastic programming, decision theory, stochastic search, simulation-optimization, stochastic control, approximate dynamic programming, and reinforcement learning. While these communities generally co-exist fairly peacefully, there are ongoing debates between the use of simulation vs. optimization, or stochastic programming vs. dynamic programming. A helpful referee for a recent paper (Powell et al. (2012*b*)) offered the advice:

One of the main contributions of the paper is the demonstration of a policy-based modeling framework for transportation problems with uncertainty. However, it could be argued that a richer modeling framework already exists (multi-stage stochastic programming) that does not require approximating the decision space with policies.

This quote highlights one of the more contentious debates in stochastic optimization within the operations research community: stochastic programming or dynamic programming? It is well known, of course, that dynamic programming suffers from the curse of dimensionality, so there is no need to learn this field if you want to work on real problems. Even I concluded this in the 1980s while looking for methods to solve stochastic fleet management problems. But 20 years later, I finally cracked these problems with successful systems that are planning driver fleets for one of the largest truckload carriers in the country (Simao et al. (2009)), and planning locomotives at one of the largest railroads in the country (Bouzaiene-Ayari et al. (2012)). The same algorithmic strategy was used to solve a stochastic energy investment problem in hourly increments over 20 years with 175,000 time periods (see Powell et al. (2012*a*)). How were these ultra high-dimensional stochastic optimization problems solved? Dynamic programming. However, an answer such as this perpetuates fundamental misconceptions about stochastic programming and dynamic programming.

As a hint to where this discussion is going, by the end of this tutorial I will have made the following points:

- Dynamic programming is a sequential (and for our purposes, stochastic) decision problem. Bellman's optimality equation is not a dynamic program; it is a) a way of characterizing an optimal policy and b) an algorithmic strategy for certain problem classes of dynamic programs.
- Stochastic programming (for multistage problems) is, with some exceptions, a lookahead policy which involves solving a lookahead model (which is itself a dynamic program) for solving a larger

dynamic program.

- The optimal solution of a multistage stochastic program is (with rare exceptions) not an optimal policy. A bound on a (multistage) stochastic program is not a bound on the quality of the policy (again, with rare exceptions).
- All properly modeled dynamic programs are Markovian. So-called "history-dependent" problems are simply dynamic programs with an incomplete state variable. If your system is not Markovian, you have not properly modeled the state variable.
- With deterministic problems, we are looking for an optimal *decision* (or vector of decisions). With (multistage) stochastic optimization problems, we are looking for optimal *functions* (known as policies).
- In my experience, every (multistage) stochastic optimization problem can be solved using one of four classes of policies (or hybrids formed from combinations of these four fundamental classes). However, optimal policies are rare.

In the process of making these points, I will bring all the fields of stochastic optimization together under a single umbrella which I suggest should be called *computational stochastic optimization*.

2 Modeling a sequential stochastic optimization problem

For many years, I was jealous of my colleagues working on deterministic integer programming problems who would present a model (difficult, but doable), and then focus on the challenging problem of designing efficient, high quality algorithms. I was working on large-scale stochastic optimization problems in transportation, but I did not enjoy the same type of roadmap to follow when writing down a model.

Several styles have evolved to model a stochastic optimization problem (keep in mind our focus on multiperiod, sequential problems). Below, we briefly describe two classical and widely used modeling styles, drawn from the fields of dynamic programming and stochastic programming. However, we are going to argue that these are not true models, but rather are closer to algorithms. We follow this discussion with a presentation of what we feel is the correct way to model a sequential decision process, using a format that is actually quite familiar in control theory.

2.1 A dynamic programming "model"

If the idea is to solve a problem with dynamic programming, many authors start by writing down the canonical form of Bellman's equation

$$V_t(S_t) = \min_{a \in \mathcal{A}} \left(C(S_t, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|S_t, a) V_{t+1}(s') \right), \tag{8}$$

where:

$$
\begin{aligned}
S_t &= \text{the state at time } t, \\
a &= \text{the (typically discrete) action in set } \mathcal{A}, \\
C(S_t, a) &= \text{the cost of being in state } S_t \text{ and taking action } a, \\
p(s'|s, a) &= \text{the probability of transitioning to state } S_{t+1} = s' \text{ if we are in state } S_t = s \text{ and take action } a, \\
V_t(s) &= \text{the value of being in state } S_t = s \text{ at time } t \text{ and following the optimal policy from } t \text{ onward.}
\end{aligned}
$$

This is the format introduced by Bellman (1957) and popularized by many authors (such as Puterman (2005)). The format is elegant and has enabled a rich theoretical tradition. A well known concept in dynamic programming is that of a *policy* which is typically denoted π, where $\pi(s)$ gives the action that should be taken when we are in state s. The policy (at time t) might be computed using

$$\pi(s) = \arg\min_{a \in \mathcal{A}} \left(C(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V_{t+1}(s') \right).$$

The most difficult challenge with Markov decision processes is that the community almost uniformly assumes that states are discrete, and that value functions and policies use lookup table representations. This sharply limits its applicability to relatively small problems.

2.2 A stochastic programming "model"

The stochastic programming community often models a stochastic programming model as follows:

$$\min_{x_t, (x_{t'}(\omega), t < t' \leq t+H), \forall \omega \in \Omega_t} \left(c_t x_t + \sum_{\omega \in \Omega_t} p(\omega) \sum_{t'=t+1}^{t+H} c_{t'}(\omega) x_{t'}(\omega) \right). \tag{9}$$

Here, ω is called a *scenario* drawn from a sampled set Ω_t generated for the problem we are solving at time t (many authors prefer "s" for scenario, but ω is widely used and avoids the conflict with standard notation for state). If we have a random sample, the probability of scenario ω might be $p(\omega) = 1/|\Omega_t|$. We make one decision x_t for time t, but then we have to make a decision $x_{t'}(\omega)$ for each scenario ω in the future. Note that we are writing the problem as if we are sitting at time t, to be consistent with our dynamic program above.

The constraints are tricky to write out. We start with the time t constraints which we might write

$$
\begin{aligned}
A_t x_t &= b_t, && (10) \\
x_t &\geq 0. && (11)
\end{aligned}
$$

We then have to write out constraints for time periods $t' > t$, which have to be written for each scenario ω. These might be written

$$A_{t'}(\omega)x_{t'}(\omega) - B_{t'-1}(\omega)x_{t'-1}(\omega) = b_{t'}(\omega), \quad (12)$$

$$x_{t'}(\omega) \geq 0. \quad (13)$$

Note that we require that $x_t = x_{t'}(\omega)$ for $t' = t$, a condition that is often imposed as a *nonanticipativity constraint*.

While this format is by far the most common, it is important to note that we have modeled the problem as if we make a single decision at time t (represented by x_t), then see a scenario ω that describes all the random information over the planning horizon $t+1, \ldots, t+H$. Once this information is revealed, we then choose $x_{t+1}, \ldots, x_{t+H}$. In reality, we choose x_t, then see the information W_{t+1}, then we would choose x_{t+1}, after which we see W_{t+2}, and so on. The model given by equations (9)-(13) is known as a two-stage approximation; it is widely used simply because the multistage version is completely intractable for most applications. In a nutshell, scenario trees have their own curse of dimensionality, which is actually worse than the one suffered by traditional dynamic programs because the entire history is captured.

2.3 The five elements of a sequential decision problem

What is known as a "dynamic programming model" (equation (8)) or the "stochastic programming model" (equations (9)-(13)) are not actually models, but are closer to algorithms. Bellman's equations, in particular, are like complementary slackness conditions for a linear program. It is not a model, but rather is similar to the optimality conditions for a dynamic program. Clearly, we need a canonical *model* for sequential ("multistage") stochastic decision problems that parallels our deterministic optimization model given by (4)-(7).

There are five fundamental elements to any sequential stochastic optimization problem. These are:

- State S_t - This represents what we know at time t before we make a decision (more on this below).
- Actions - Depending on the community, these might be discrete actions a, continuous controls u, or decisions x (which are typically vectors, and might be continuous, integer or a mixture). We defer to later the choice of how a decision is made, but assume that we will design a *policy* π. Dynamic programmers will write a policy as $\pi(s)$ returning an action a (or u or x), but this leaves open the question of what $\pi(s)$ looks like computationally. For this reason, we adopt a different notation. If our decision is action a_t, we designate the policy as the function $A_t^\pi(S_t)$. If we are using decision x_t, we use the function $X_t^\pi(S_t)$. We assume that our policy produces feasible actions (say, $x_t \in \mathcal{X}_t$), where the feasible region $\mathcal{X}_t$ might be a system of linear equations that depends on the state S_t.

- Exogenous information W_t - Starting at time 0, we observe exogenous information (prices, demands, equipment breakdowns) as the sequence $W_1, W_2, \ldots$. We use the convention that any variable indexed by t is known at time t. This means that states, actions (decisions) and information evolve as follows:

$$(S_0, x_0, W_1, S_1, x_1, W_2, \ldots, S_t, x_t, W_{t+1}, \ldots, S_T).$$

An important concept is the *post-decision state* which we write as S_t^x (or S_t^a if we are using action a), which is the information in the system immediately *after* we make a decision (see Powell (2011)[Chapter 4] for a thorough discussion of post-decision state variables). Introducing the post-decision state in our sequence gives us

$$(S_0, x_0, S_0^x, W_1, S_1, x_1, S_1^x, W_2, \ldots, S_t, x_t, S_t^x, W_{t+1}, \ldots, S_T).$$

If the information process evolves exogenously, independently of current states and actions (true for many, but not all, problems), it is convenient to let ω be a sample realization of the random variables $W_1, W_2, \ldots, W_T$. We then let Ω be the set of all possible realizations. For some models, it is useful to represent the *history* of the process h_t at time t as

$$h_t = (W_1, \ldots, W_t). \tag{14}$$

To allow for the possibility that states and/or actions do affect the information process, some will model ω as the sequence of states and actions (referred to as the *induced* stochastic process in Puterman (2005)), or the sequence of initial state, followed by all $x_{t'}$ and $W_{t'}$ for $t' = 1, \ldots, t$. Our presentation assumes the information process is purely exogenous, so we use the representation of history in (14).

- The transition function - We write this as $S_{t+1} = S^M(S_t, x_t, W_{t+1})$. The transition function may include (controllable) systems of linear equations such as those shown in equation (6). However, it may also represent the exogenous evolution of prices, weather, and customer demands. For example, let p_t be the price of electricity, and let $\hat{p}_{t+1}$ be the exogenous change in the price between t and $t+1$ ($\hat{p}_t$ would be an element of W_t), we would write

$$p_{t+1} = p_t + \hat{p}_{t+1}.$$

This would be an example of a transition function for an exogenous information process such as prices.

- The objective function - We assume we are given a cost/reward/utility function that we refer to generically as the *contribution function* which may depend on the state S_t and the action x_t, so we write it as $C(S_t, x_t)$. In some settings, it also depends on the new information W_{t+1}, in which case we would write it as $C(S_t, x_t, W_{t+1})$, which means it is random at time t. The engineering controls community and the computer science community often deal with "model-free" problems where the transition function is unknown and W_{t+1} is not observable, but where

the cost depends on the downstream state S_{t+1}, in which case the contribution would be written $C(S_t, x_t, S_{t+1})$, which is also random at time t.

The objective requires finding the policy that minimizes expected costs, which is written

$$\min_{\pi \in \Pi} \mathbb{E}^{\pi} \sum_{t=0}^{T} C(S_t, X_t^{\pi}(S_t)), \tag{15}$$

where $S_{t+1} = S^M(S_t, x_t, W_{t+1})$, and where the expectation is over all possible sequences $W_1, W_2, \ldots, W_T$, which may depend on the actions taken. The notation $\mathbb{E}^{\pi}$ allows for the possibility that the exogenous information might depend on prior actions (something that is not allowed in traditional stochastic programming models). The goal here is to find the best policy, which means that we are looking for the best *function* for making a decision.

This leaves open the computational question: How in the world do we search over some abstract space of policies? Answering this question is at the heart of this chapter. We are going to argue that different communities (dynamic programming, stochastic programming, simulation) get around this problem by adopting a specific class of policies.

There is growing interest in replacing the expectation with various risk measures, which introduces the issue of non-additivity (see Shapiro et al. (2009), Ruszczynski (2010) for thorough discussions). For example, we might replace (15) with

$$\min_{\pi \in \Pi} Q_{\alpha} \sum_{t=0}^{T} C(S_t, X_t^{\pi}(S_t)), \tag{16}$$

where $Q_{\alpha}(W)$ is the α-quantile of the random variable W. Alternatively, we might use a *robust* objective which uses

$$\min_{\pi \in \Pi} \max_{W \in \mathcal{W}} \sum_{t=0}^{T} C(S_t, X_t^{\pi}(S_t)), \tag{17}$$

where the maximum over $W \in \mathcal{W}$ refers to a search over random variables within an uncertainty set $\mathcal{W}$ (Bandi & Bertsimas (2012)) that defines a set of "worst case" values. See Beyer & Sendhoff (2007), Ben-Tal et al. (2009) and for further discussions of robust optimization.

It is useful to stop and compare our stochastic model with our deterministic model, which we gave previously as (4)-(7). Our stochastic objective function, given by equation (15), should be compared to its deterministic counterpart, equation (4). The linking equations (6) can be written:

$$A_t x_t = R_t, \tag{18}$$

$$R_t = b_t + B_{t-1} x_{t-1}. \tag{19}$$

If we let $S_t = R_t$, we see that equation (6), in the form of equations (18)-(19), represents the transition function $S_t = S^M(S_{t-1}, x_{t-1}, \cdot)$. Obviously, there is no deterministic counterpart to the exogenous information process.

This style of modeling problems has not been used in operations research (or computer science), but is widely used in the engineering controls community. The dynamic programming community prefers to assume that the one-step transition matrix (used in equation (8)) is given. While it can, in principle, be computed from the transition function and distributional information on the exogenous information variable W_t, it is simply not computable for the vast majority of applications. The transition function is widely used in stochastic programming, but without using the term or the notation (for those familiar with stochastic programming, this is how scenario trees are built).

This model brings out the fundamental distinction between deterministic and stochastic optimization (for sequential decision problems). In deterministic optimization, we are looking for the best decisions (see (4) - (7)). In stochastic optimization, we are looking for the best policy (see (15)), which is the same as searching over a class of functions. The operations research community is quite skilled at solving for high-dimensional vectors of decision variables, but searching over functions is a less familiar concept. This is going to need some work.

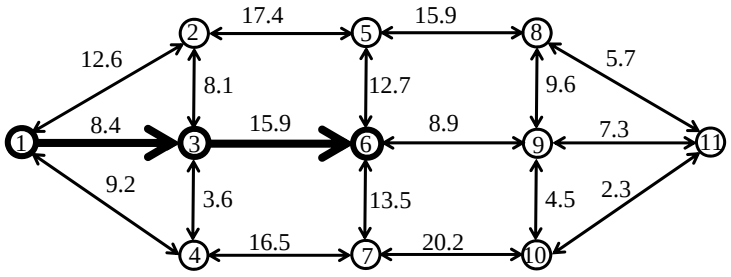
The two dimensions of our modelling framework that are least understood are the state variable, and the concept of searching over policies in the objective function. For this reason, we deal with these topics in more depth.

3 What is a state variable?

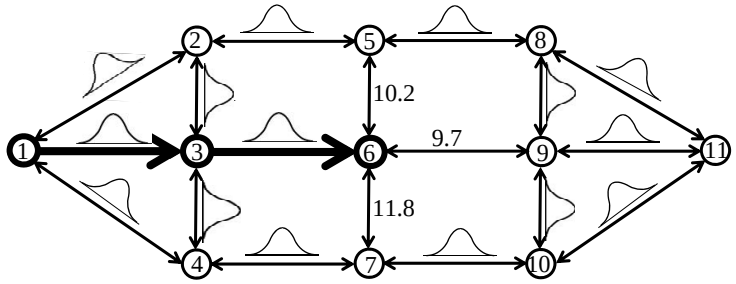
State variables appear to be a concept that everyone understands, but cannot provide a definition. Bellman (1957) offered "... we have a physical system characterized at any stage by a small set of parameters, the *state variables*." Puterman (2005) introduces state variables with "At each decision epoch, the system occupies a *state*." (Italics are in the original manuscripts). Wikipedia (in 2013) suggests "State commonly refers to either the present condition of a system or entity," which is true but hopelessly vague, or "A state variable is one of the set of variables that are used to describe the mathematical state of a dynamical system" (using the word you are trying to define in the definition is a red flag that you do not have a definition). Other top experts have suggested that the state variable cannot be defined, or they say they have simply given up. I think we can do better.

The graphs in figure 1 help to illustrate the definition of a state variable. In figure 1(a), we show a deterministic graph where a traveler has to get from node 1 to node 11. The traveler is currently at node 6. Let N_t be the node number of the traveler after t link traversals ($t = 2$ in the example). There seems to be a consensus that the correct answer is

$$S_t = N_t = 6.$$



(a)



(b)

Figure 1: (a) Deterministic network for a traveler moving from node 1 to node 11 with known arc costs. (b) Stochastic network, where arc costs are revealed as the traveler arrives to a node.

Note that we exclude from our state variable all data that is not changing (the arc costs are all assumed to be deterministic).

Next assume that we have a stochastic graph, where the probability distribution of the cost of traversing each arc is known, as depicted in figure 1(b). However, assume that if the traveler arrives to node i, he is able to see the actual cost $\hat{c}_{ij}$ for each link (i, j) out of node i. Again, there seems to be broad agreement that the correct answer is

$$S_t = (N_t, (\hat{c}_{N_t,\cdot})) = (6, (10.2, 9.7, 11.8)),$$

where $(\hat{c}_{N_t,\cdot})$ represents the costs on all the links out of node N_t. While our first example illustrates a physical state (the location of our traveler), this second example illustrates that the state also includes the state of information.

For our final example, we introduce the twist that there are left hand turn penalties for our stochastic network in figure 1(b). Now what is the state variable? In the previous example, we saw that we need to include all the information we need to make a decision. In this example, we need to know the previous node, that is, N_{t-1}. With this information, we can tell if the link from our current node $N_t = 6$ to node 5 is a left hand turn or not. So, our state variable in this setting would be

$$S_t = (N_t, (\hat{c}_{N_t,\cdot}), N_{t-1}) = (6, (10.2, 9.7, 11.8), 3).$$

If we need N_t and N_{t-1}, what about the other nodes we have visited? For the same reason that we did not include the complete path in our first example, we do not need node N_{t-2} (and earlier) in this example simply because they are not needed. This situation would change if we introduced a requirement that our path contain no cycles, in which case we would have to retain the entire path as we progress.

There is an astonishing lack of agreement in the definition of a state variable. This question was posed to a workshop of attendees from computer science and operations research. Two themes emerged from the responses. The first (popular with the computer scientists) was that a state variable should be a *sufficient statistic*, a term drawn from the statistics community (which has its own definition for this term). The second was that a state variable should be parsimonious. The concepts of sufficiency and parsimony (said differently, necessary and sufficient), are completely consistent with the state descriptions given above.

I would conclude from all of these discussions that a state variable should include all the information we need, but only the information we need. But to do what? In Powell (2007), I offer the definition:

Definition 3.1 *A* **state variable** *is the minimally dimensioned function of history that is necessary and sufficient to compute the decision function, the transition function, and the contribution function.*

It is useful to divide state variables into three broad classes:

- Physical (or resource state) R_t - This might be the node where we are located on a graph, the amount of product in inventory, or the speed of an aircraft.
- Information state I_t - This includes other information such as information about weather, the costs out of a node, or the elapsed time since the last customer arrived to a queue.
- The knowledge (or belief) state K_t - This comes in the form of a series of distributions about unobservable parameters. For example, this might be the demand for a product at a particular price, or the travel time on a path through an unfamiliar city. The knowledge state arises in communities that use names such as partially observable Markov decision processes (POMDPs), multi-armed bandit problems (Gittins et al. (2011)), and optimal learning (Powell & Ryzhov (2012)).

Mathematically, the information state I_t should include information about resources R_t, and the knowledge state K_t should include everything, as illustrated in figure 2. However, in terms of practical models, it helps to divide variables among the three categories, primarily because so many people fall in the trap of equating "state" with "physical state." Whether a variable is listed under "physical state variables" or "information state variables" is not important. It is useful to separate out the knowledge state, which comes in the form of a probability distribution.

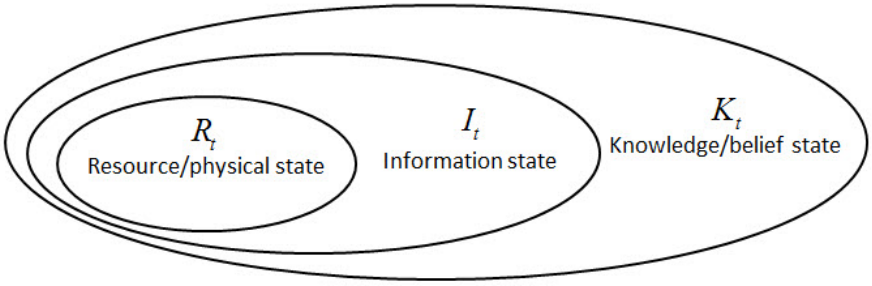


Figure 2: Illustration of the growing sets of state variables, where information state includes physical state variables, while the knowledge state includes everything.

Another way to envision the state variable is summing up the contributions while following some policy, which we might write as

$$F_0^\pi(S_0) = \mathbb{E}\left\{\sum_{t'=0}^{T} C(S_{t'}, X_{t'}^\pi(S_{t'}))\middle| S_0\right\},$$

which is computed given our initial state S_0. We assume states are related by $S_{t+1} = S^M(S_t, X_t^\pi(S_t), W_{t+1})$, and the sequence $W_1, W_2, \ldots, W_T$ follows a sample path $\omega \in \Omega$. Now assume we want to do the same computation starting at time t, which we would write

$$F_t^\pi(S_t) = \mathbb{E}\left\{\sum_{t'=t}^{T} C(S_{t'}, X_{t'}^\pi(S_{t'}))\middle| S_t\right\}. \tag{20}$$

We can, of course, write this as

$$\begin{aligned} F_t^\pi(S_t) &= C(S_t, X_t^\pi(S_t)) + \mathbb{E}\left\{\sum_{t'=t+1}^{T} C(S_{t'}, X_{t'}^\pi(S_{t'}))\middle| S_t\right\} &(21)\\ &= C(S_t, X_t^\pi(S_t)) + \mathbb{E}\{F_{t+1}^\pi(S_{t+1})|S_t\}. &(22)\end{aligned}$$

Now it is apparent that S_t (or in the case of (22), S_{t+1}) has to carry all the information needed to compute $F_t(S_t)$ (or $F_{t+1}(S_{t+1})$). We have to include in S_t all the information needed to compute the policy $X_{t'}^\pi(S_{t'})$, the contribution function $C(S_{t'}, X_{t'}^\pi(S_{t'}))$, and the transition function $S^M(S_{t'}, X_{t'}^\pi(S_{t'}), W_{t'+1})$, for all $t' = t, ..., T$. Not surprisingly, we see no need to include any information in S_t that is not needed to compute any of these functions.

While the principles behind this definition seem to have broad support, they carry implications that run against conventional thinking in the operations research community. First, there is no such thing as a non-Markovian system, because any properly modeled state variable includes all the information needed to model the forward trajectory of the system (yes, even $G/G/1$ queues are

Markovian when properly modeled). Second, the oft-repeated statement that "any system can be made Markovian by adding enough variables" needs to be replaced with the response "if your system is not Markovian, you do not have a complete state variable, and if you can add information to make the system Markovian, then you should!"

It is time to teach our students what a state variable is, so that they learn how to properly model dynamic systems.

4 Designing policies

Far more difficult than understanding a state variable is understanding what in the world we mean by "searching for a policy." This is the type of statement that is easy to say mathematically, but seems on the surface to have no basis in practical computation. Perhaps this was the reaction to Kantarovich's statement of a linear program, and the reason why linear programming became so exciting after Dantzig introduced the simplex algorithm. Dantzig made Kantarovich's linear program meaningful.

First, we need a definition of a policy:

Definition 4.1 *A* **policy** *is a mapping from a state to a feasible action. Any mapping.*

Mathematically, we can think of an arbitrarily complex array of functions, but no-one knows how to calculate these functions. Imagine, for example, that our action is a vector x that might include thousands of continuous and integer variables that have to satisfy a large set of linear constraints. How are we going to find a function that accomplishes this? This section is devoted to this question.

4.1 The four classes of policies

I would argue that rather than devise some dramatic breakthrough in functional approximations, all we have to do is to look at the wide array of tools that have already been used in different applications. In my own tour through the jungle of stochastic optimization, I have found four fundamental classes of policies which I call PFAs, CFAs, VFAs, and Lookahead policies.

- Policy function approximations (PFAs) - A policy function approximation represents some analytic function that does not involve solving an optimization problem. A PFA might be a lookup table ("turn left at a particular intersection" or "move the knight when the chessboard is in a particular state"), a rule ("sell if the price goes above θ"), or a parametric model such as

$$X^\pi(S_t|\theta) = \theta_0 + \theta_1 S_t + \theta_2 S_t^2. \tag{23}$$

Another example of a PFA is an (s, S) inventory model: order product when the inventory is below s to bring it up to S. The engineering controls community often uses neural networks to represent a policy. Many simulation models contain imbedded decisions (e.g. how to route a job in a jobshop) that are governed by simple rules (we would call these policy function approximations).

- Optimizing a cost function approximation (CFAs) - Here, we replace the cost function (this could be the cost over a single time period, or over a planning horizon H) with an approximation that we call $\bar{C}_t^\pi(S_t, x|\theta)$, giving us the policy

$$X_t^\pi(S_t|\theta) = \arg\min_{x\in\mathcal{X}_t} \bar{C}^\pi(S_t, x|\theta). \tag{24}$$

The vector θ represents tunable parameters that might appear as modifications to the cost coefficients, right hand side coefficients, or even the coefficients of the constraint matrix itself. We let π capture both the structure of our modified cost function as well as the tunable parameters, but we express the tunable parameters θ explicitly. A special case of a CFA policy is where $\bar{C}^\pi(S_t, x|\theta) = C(S_t, x)$, in which case we would have a simple myopic policy. Another special case uses a cost function of the form

$$X_t^\pi(S_t|\theta) = \arg\min_{x\in\mathcal{X}_t} \big(C(S_t, x) + \sum_{f\in\mathcal{F}} \theta_f \phi_f(S_t, x)\big). \tag{25}$$

Here, $(\phi_f(S_t, x))$, $f \in \mathcal{F}$, is a set of *basis functions* (as they are known in the approximate dynamic programming community) which might be of the form $S_t x$, $S_t x^2$, x, x^2, which serves as a form of correction function (more on this below).

- Policies that depend on a value function approximation (VFAs) - These are the policies most often associated with dynamic programming, and are written

$$X_t^\pi(S_t|\theta) = \arg\min_{x\in\mathcal{X}_t} \big(C(S_t, x) + \mathbb{E}\{\overline{V}_{t+1}(S_{t+1}|\theta)|S_t\}\big). \tag{26}$$

where $S_{t+1} = S^M(S_t, x_t, W_{t+1})$. Since expectations can be notoriously hard to compute (imagine if our random variable W_t has, say, 100 dimensions), we can use the device of the post-decision state variable (the state immediately after a decision has been made, but before any new information has arrived). Let S_t^x be the post-decision state, which means it is a deterministic function of x_t. This allows us to write

$$X_t^\pi(S_t|\theta) = \arg\min_{x\in\mathcal{X}_t} \big(C(S_t, x) + \overline{V}_t(S_t^x|\theta)\big). \tag{27}$$

In both (26) and (27), we have to create an approximation of the value function. Again, we assume that the index π captures the structure of the function, as well as any tunable parameters represented by θ. For example, a popular approximation is linear regression. Assume that

someone has devised a series of explanatory variables ("basis functions") $\phi_f(S_t^x)$ for $f \in \mathcal{F}$. Then we can write

$$X_t^\pi(S_t|\theta) = \arg\min_{x\in\mathcal{X}_t} \big(C(S_t,x) + \sum_{f\in\mathcal{F}} \theta_f \phi_f(S_t^x)\big). \tag{28}$$

Here, the index π carries the information that we are using a linear architecture for the value function, the set of basis functions, as well as the coefficients θ used in the linear model. We note that while (25) and (28) look similar, the mechanisms for fitting the regression coefficients are completely different. In (28), we are trying to approximate the future contributions given that we are in state S_t; in (25), we are just computing an adjustment term which is unlikely to bear any relationship to future contributions.

- Lookahead policies - The simplest lookahead policy involves optimizing over a horizon H deterministically. Let $\tilde{x}_{tt'}$ represent the decision variables (this might be a vector) for time t' in the lookahead model that has been triggered at time t. A deterministic lookahead policy might be written

$$X_t^\pi(S_t|\theta) = \arg\min_{\tilde{x}_{tt},\ldots,\tilde{x}_{t,t+H}} \sum_{t'=t}^{t+H} C(\tilde{S}_{tt'},\tilde{x}_{tt'}). \tag{29}$$

Normally, we optimize over the horizon $(t,\ldots,t+H)$ but only implement the first decision, so $X_t^\pi(S_t|\theta) = \tilde{x}_{tt}$. All variables in the lookahead model are indexed by (tt') where t represents when the decision is being made (and therefore its information content), while t' is when the decision impacts the physical system. We also use tilde's to avoid confusion between the lookahead model (which often uses a variety of approximations) and the real model. Here, the parameter θ might represent the horizon H.

We might also use a stochastic lookahead model

$$\min_{\tilde{x}_{tt},(\tilde{x}_{tt'}(\tilde{\omega}),t<t'\leq t+H),\forall\tilde{\omega}\in\tilde{\Omega}_t} \left(\tilde{c}_{tt}\tilde{x}_{tt} + \sum_{\tilde{\omega}\in\tilde{\Omega}_t} p(\tilde{\omega}) \sum_{t'=t+1}^{t+H} \tilde{c}_{tt'}(\tilde{\omega})\tilde{x}_{tt'}(\tilde{\omega})\right). \tag{30}$$

In this case, θ captures parameters such as the number of information stages (two in this example) and the number of samples per stage (this community refers to the elements of $\tilde{\Omega}_t$ as *scenarios*. We need to construct a lookahead stochastic process, captured by the set $\tilde{\Omega}_t$, to differentiate stochastic outcomes in the lookahead model from the sample path $\tilde{\omega} \in \tilde{\Omega}_t$ that we might be following in the real model. The choice of the number of stages, and the construction of the set $\tilde{\Omega}_t$, represent critical choices in the design of the lookahead model, which we parameterize by θ.

We note that a stochastic lookahead model is the strategy favored by the stochastic programming community. Equation (30) can be described as a *direct* solution of the lookahead model, which is to say that we explicitly optimize over all decisions, for each sample path (scenario)

ω, all at the same time. It is also possible to solve the lookahead model using value functions, producing a policy that looks like

$$X_t^\pi(S_t|\theta) = \arg\min_{x_{tt}\in\mathcal{X}_t} \big(C(S_{tt}, x_{tt}) + \mathbb{E}\{\tilde{V}_{t,t+1}(\tilde{S}_{t,t+1})|S_t\}\big), \tag{31}$$

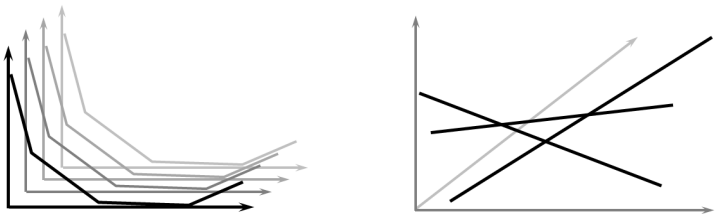
where $\tilde{V}_{t,t+1}(\tilde{S}_{t,t+1})$ might be an exact (or nearly exact) estimate of the value of being in state $\tilde{S}_{t,t+1}$ of the (approximate) lookahead model. We emphasize that lookahead models are almost always approximations of the true model; we might use aggregation, discretization, and/or Monte Carlo sampling along with a limited horizon to reduce the true model to something tractable. For sequential convex optimization problems, it is possible to use Benders cuts, a strategy that has become known as *stochastic dual decomposition programming* (SDDP) (see Pereira & Pinto (1991) and Shapiro et al. (2013)).

In addition to these four fundamental classes of policies, we can also create hybrids by mixing and matching. For example, we might use a lookahead policy with a value function at the end of the lookahead horizon. Another powerful strategy is to combine a low-dimensional policy function approximation (say, "maintain 10 units of Type A blood in inventory") as a goal in a larger, higher-dimensional optimization problem (see Defourny et al. (2013)). Cost function approximations, which include any modification of costs or constraints to achieve a more robust policy, is often combined with a lookahead policy so that forecasts can be accommodated.

4.2 Approximating functions

Three of our four policies require some sort of functional approximation: policy function approximations, cost function approximations, and value function approximations. There is a wide range of methods for approximating functions, although the most popular can be divided into three classes: lookup tables, parametric, and nonparametric (see Hastie et al. (2009) for a thorough review of statistical learning methods). For example, in the field of Markov decision processes, the use of policies based on value functions represented by lookup tables is not an approximation under ideal circumstances. However, since lookup table representations do not scale, most applications require some sort of parametric model (not necessarily linear). Indeed, the habit of equating "Markov decision processes" with lookup table representations of value functions is the reason why so many have dismissed "dynamic programming" because of the curse of dimensionality. However, there is absolutely no need to insist on using lookup tables, and this has made it possible for approximate dynamic programming to produce practical solutions to some truly large-scale applications (e.g. Simao et al. (2009) and Bouzaiene-Ayari et al. (2012)).

An important class of problems in operations research involves convex optimization models, which frequently arise in the context of resource allocation. Figure 3 illustrates two (nonparametric) methods for approximating convex functions: piecewise linear, separable approximations (a) (see



(a) (b)

Figure 3: Approximating convex functions using piecewise linear, separable approximations (a), and multidimensional Benders cuts (b).

Powell (2011)[Chapter 13], and multidimensional Benders cuts (b) (see Shapiro et al. (2009) and Birge & Louveaux (2011)). An extensive literature on exploiting these approximation methods for resource allocation problems is beyond the scope of our discussion.

4.3 Evaluating a policy

It would be nice if we could simply compute the objective function in equation (15), but situations where we can compute the expectation exactly are quite rare. For this reason, we generally depend on using Monte Carlo simulation to get a statistical estimate of the value of a policy. Let ω represent a sample realization of the sequence $(W_1, W_2, \ldots, W_T)$. A single simulation of a policy would be written

$$F^{\pi}(\omega) = \sum_{t=0}^{T} C(S_t(\omega), X_t^{\pi}(S_t(\omega))).$$

If T is large enough, we might feel that this is a reasonable estimate of the value of a policy π, but more often we are going to compute an average using

$$\bar{F}^{\pi} = \frac{1}{N} \sum_{n=0}^{N} F^{\pi}(\omega^n). \tag{32}$$

If we are simulating a policy based on value function approximations, we might write our VFA as

$$\overline{V}_t(S_t|\theta) = \sum_{f \in \mathcal{F}} \theta_f \phi_f(S_t).$$

There are different strategies for estimating the regression coefficients θ. If we are using adaptive learning (common in approximate dynamic programming), the policy at iteration n would be given

by

$$X_t^\pi(S_t|\theta^{n-1}) = \arg\min_{x_t}\left(C(S_t,x_t) + \sum_{f\in\mathcal{F}}\theta_f^{n-1}\phi_f(S_t)\right).$$

Note that during iteration n, we might use estimates θ^{n-1} obtained from iterations $n-1$ and earlier.

4.4 Searching for the best policy

We can now put meaning to the statement "search over policies" (or "find the best function") implied by the objective function in equation (15). The label π on our policy $X_t^\pi(S_t)$ carries two types of information:

Categorical information which describes the type of policy (PFA, CFA, VFA and Lookahead), and would also have to specify the specific structure of a function approximation: lookup table, parametric, or nonparametric (for example). If parametric (which is the most common), the categorical information would have to describe the particular structure (e.g. the set of basis functions in a linear approximation).

Tunable parameters that we refer to as θ, which might be the regression parameters of a linear model, the planning horizon in a lookahead policy, and the number of samples in a scenario tree.

Let $C \in \mathcal{C}$ represent a class of policies, and let Θ^C be the set of tunable parameters θ for that class. So, $\pi^C = (C,\theta)$ for $C \in \mathcal{C}$ and $\theta \in \Theta^C$. For a particular class of policy $C \in \mathcal{C}$, we have to solve a stochastic search problem to find the best $\theta \in \Theta^C$, which we write as

$$\min_{\theta\in\Theta^C} \bar{F}^{\pi^C}(\theta) = \frac{1}{N}\sum_{t=0}^{T} C(S_t(\omega^n), X_t^{\pi^C}(S_t(\omega^n)|\theta)). \tag{33}$$

This problem has to be solved for each of a (hopefully small) class of policies C chosen by the designer.

The literature for solving stochastic search problems is quite deep, spanning stochastic search (e.g. Spall (2003), Chang et al. (2007)), two-stage stochastic programming (Birge & Louveaux (2011), Shapiro et al. (2009), Kleywegt et al. (2002)), simulation-optimization (Swisher et al. (2000), Fu et al. (2005), Hong & Nelson (2007), Chick et al. (2007), Chick & Gans (2009), Andradóttir & Prudius (2010)), ranking and selection (Barr & Rizvi (1966), Boesel et al. (2003)), and optimal learning (see the review of many techniques in Powell & Ryzhov (2012)). Algorithms vary depending on the answers to the following questions (to name a few):

- Is the objective function convex in θ?

- Can we compute derivatives (for continuous, tunable parameters)?
- Are we simulating policies in the computer (offline), or are we observing policies as they are being used in the field (online)?
- Can the objective function be quickly computed, or is it time consuming and/or expensive?
- Is the dimensionality of θ small (three or less), or larger?

5 Lookahead policies

By far the most complex policy to model is a lookahead policy, which solves the original model by building a series of approximate models known as lookahead models. A lookahead model to be solved at time t is typically formulated over a horizon $t, \ldots, t+H$, and is used to determine what to do at time t, given by x_t (or a_t or u_t). Once we implement x_t, we observe the transition from state S_t to S_{t+1} and repeat the process. Lookahead policies are often referred to as rolling/receding horizon procedures, or model predictive control. It is not unusual for authors to formulate a lookahead model without actually writing down the true model.

But not everyone is using a lookahead model, which means it is not always clear whether a paper is solving the true model or a lookahead model. Imagine that we are trying to model a business problem over a horizon spanning time periods 0 up to 100. Is the intent to determine what to do at time 0? Or are we modeling a system over a planning horizon to assess the impact of various parameters and business rules? If we have a business simulator, then we will need to make decisions at every time t within our simulation horizon $t = 0, \ldots, T$, which we typically need to repeat for different sample outcomes. In a lookahead model, we also need to make decisions over our planning horizon $t' = t, \ldots, t+H$, but the decisions we make at time periods $t' > t$ are purely for the purpose of making a better decision at time t.

5.1 An optimal policy using the original model

We begin our discussion by noting that we can characterize an optimal policy using the function

$$X_t^*(S_t) = \arg\min_{x_t} \left(C(S_t, x_t) + \min_{\pi \in \Pi} \mathbb{E} \left\{ \sum_{t'=t+1}^{T} C(S_{t'}, X_{t'}^{\pi}(S_{t'})) \middle| S_t \right\} \right), \tag{34}$$

where $S_{t+1} = S^M(S_t, x_t, W_{t+1})$. The imbedded optimization over policies can look mysterious, so many use an equivalent formulation which is written

$$X_t^*(S_t) = \arg\min_{x_t} \left(C(S_t, x_t) + \min_{(x_{t'}(\omega), t < t' \leq t+H), \forall \omega \in \Omega_t} \mathbb{E} \left\{ \sum_{t'=t+1}^{T} C(S_{t'}, x_{t'}(\omega)) \middle| S_t \right\} \right). \tag{35}$$

Instead of writing a policy $X^\pi_{t'}(S_{t'})$, we are writing $x_{t'}(\omega)$. To make this valid, we have to impose a condition that means that $x_{t'}(\omega)$ cannot "see" into the future (a condition that is satisfied when we use our policy $X^\pi_{t'}(S_{t'})$), since specifying the outcome ω implies we know the entire future. One way to do this is to write $x_{t'}(h_{t'})$, expressing the dependence of $x_{t'}$ on the history of the information process (see equation (14)). The problem is that since $h_{t'}$ is typically continuous (and multidimensional), $x_{t'}(h_{t'})$ is effectively a continuous function which is the decision we would make if our history is $h_{t'}$ (in other words, a policy).

Both (34) and (35) are lookahead policies that use the original model. Both require computing not only the decision that determines what we do now (at time t), but also policies for every time period in the future. The problem, of course, is that we cannot compute any of these policies, leading us to consider approximate lookahead models.

5.2 Building an approximate lookahead model

To overcome the complexity of solving the exact model, we create what is called a *lookahead model* which is an approximation of the original model which is easier to solve. To distinguish our lookahead model from the real model, we are going to put tildes on all the variables. In addition, we use two time indices. Thus, the decision $\tilde{x}_{tt'}$ is a decision determined while solving the lookahead model at time t, with a decision that will be implemented at time t' within the lookahead model.

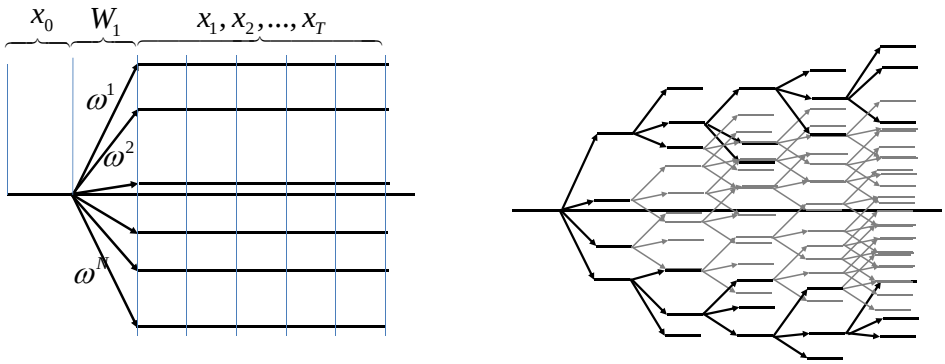
There are several strategies that are typically used to simplify lookahead models:

Limiting the horizon - 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 might be enough to produce high quality decisions.

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 (see Puterman (2005)). 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 $t + 1$.

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^n_t during the nth simulation through the horizon) through the end of our horizon $t + H$. We refer to this as $\tilde{\Omega}^n_t$ to capture that it is constructed for the decision problem at time t while in state S^n_t. The simplest model in this class is a deterministic lookahead, which uses a single point estimate.

Stage aggregation - A stage represents the process of revealing information before making another decision. A common approximation is a two-stage formulation (see figure 4(a)), where we make



(a) (b)

Figure 4: Illustration of (a) a two-stage scenario tree and (b) a multistage scenario tree.

a decision x_t, then observe all future events (until $t+H$), and then make all remaining decisions. A more accurate formulation is a multistage model, depicted in figure 4(b), but these can be computationally very expensive.

Dimensionality reduction - 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. While we have to track these in the original 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*).

5.3 A deterministic lookahead model

A simple deterministic lookahead model simply uses point forecasts of all exogenous variables, giving us

$$X_t^{LA-D,n}(S_t) = \arg\min_{x_t} \left(C(S_t, x_t) + \sum_{t'=t+1}^{t+H} C(\tilde{S}_{tt'}, \tilde{x}_{tt'}) \right), \tag{36}$$

where $\tilde{S}_{t,t'+1} = S^M(\tilde{S}_{tt'}, \tilde{x}_{tt'}, \overline{W}_{tt'})$, and where $\overline{W}_{tt'} = \mathbb{E}\{W_{tt'}|S_t\}$ is a forecast of $W_{t'}$ made at time t. Deterministic lookahead policies go under names such as rolling horizon procedures or model predictive control, and are widely used in practice.

5.4 A stochastic lookahead model

A stochastic lookahead model can be created using our sampled set of outcomes $\tilde{\Omega}_t^n$, giving us a stochastic lookahead policy

$$X_t^{LA-SP,n}(S_t^n) = \arg\min_{x_t} \left(C(S_t^n, x_t) + \min_{(\tilde{x}_{tt'}(\tilde{\omega}), t<t'\leq t+H), \forall \tilde{\omega} \in \tilde{\Omega}_t^n} \tilde{\mathbb{E}}^n \left\{ \sum_{t'=t+1}^{t+H} C(\tilde{S}_{tt'}, \tilde{x}_{tt'}(\tilde{\omega})) \middle| S_t \right\} \right). \quad (37)$$

When computing this policy, we start in a particular state S_t^n (in the state space of the original model), but then step forward in time using

$$\begin{aligned} \tilde{S}_{t,t+1} &= S^M(S_t^n, x_t, \tilde{W}_{t,t+1}(\tilde{\omega})), \\ \tilde{S}_{t,t'+1} &= S^M(\tilde{S}_{tt'}, \tilde{x}_{tt'}, \tilde{W}_{t,t'+1}(\tilde{\omega})),\ t' = t+1, \ldots, T-1. \end{aligned}$$

In (37), the expectation $\tilde{\mathbb{E}}^n \{\cdot | S_t\}$ is over the sampled outcomes in $\tilde{\Omega}_t^n$ which is constructed given that we are in state S_t^n. To help visualize these transitions, it is often the case that we have a resource variable $\tilde{R}_{tt'} = (\tilde{R}_{tt'i})_{i \in \mathcal{I}}$ (e.g. how many units of blood we have on hand of type i), where we would typically write the transition as

$$\tilde{R}_{t,t'+1}(\tilde{\omega}) = \tilde{A}_{tt'} \tilde{x}_{tt'}(\tilde{\omega}) + \hat{R}_{t,t'+1}(\tilde{\omega}),$$

where $\hat{R}_{t,t'+1}(\tilde{\omega})$ represents a sample realization of blood donations between t' and $t'+1$ in our lookahead model. In addition, we might have one or more information variables $\tilde{I}_{tt'}$, such as temperature (in an energy problem), or a market price. These might evolve according to

$$\tilde{I}_{t,t'+1}(\tilde{\omega}) = \tilde{I}_{tt'}(\tilde{\omega}) + \hat{I}_{t,t'+1}(\tilde{\omega}),$$

where $\hat{I}_{t,t'+1}(\tilde{\omega})$ is a sample realization of the change in our information variables. The evolution of the information variables is captured in the scenario tree (figure 4(a) or 4(b)), while that of the resource variables is captured in the constraints for $\tilde{x}_{tt'}$.

Now we have an expression that is computable, although it might be computationally expensive. Indeed, it is often the case that the optimization problem in (37) is so large that we have to turn to decomposition methods which solve the problem within a tolerance (see Rockafellar & Wets (1991), Birge & Louveaux (2011)). However, our notation now makes it clear that we are solving a lookahead model. Bounds on the performance of a lookahead model do not translate to bounds on the performance of the policy in terms of its ability to minimize the original objective function (15).

An alternative approach is to approximate the future using a value function approximation, giving us a VFA-based policy that looks like

$$X_t^{VFA,n}(S_t) = \arg\min_{x_t} \big(C(S_t, x_t) + \mathbb{E}\{\overline{V}_{t+1}^{n-1}(S_{t+1}) | S_t\} \big), \quad (38)$$

where

$$\overline{V}_{t+1}^{n-1}(S_{t+1}) \approx \min_{\pi\in\Pi} \mathbb{E}\left\{ \sum_{t'=t+1}^{T} C(S_{t'}, X_{t'}^{\pi}(S_{t'})) \middle| S_t \right\}.$$

Here, we are using some sort of functional approximation $\overline{V}_{t+1}^{n-1}(S_{t+1})$ which has been estimated using information collected during earlier iterations (that is why it is indexed $n-1$). What is important here is that we are approximating the real model, not the lookahead model. If we have access to a convenient post-decision state S_t^x, we can drop the expectation and use

$$X_t^{VFA,n}(S_t) = \arg\min_{x_t} \big(C(S_t, x_t) + \overline{V}_t^{n-1}(S_t^x)\big). \tag{39}$$

where the post-decision value function approximation (39) is different than the pre-decision value function approximation in (38).

We note that once we have the value function approximations $\overline{V}_t(S_t)$ (or $\overline{V}_t(S_t^x)$) for $t = 0, \ldots, T$, we have a complete policy for the original model, where we assume that the approximation $\overline{V}_t(S_t)$ gives us a value for every possible state. By contrast, the lookahead policy $X_t^{LA-SP,n}(S_t)$ works only for a particular state S_t at time t. In the process of solving the stochastic lookahead policy, we produce a series of decisions $\tilde{x}_{tt'}(\tilde{S}_{tt'})$ that can be interpreted as a policy within the lookahead model. But because it is only for a small sample of states $\tilde{S}_{tt'}$, this policy cannot be used again as we step forward in time. As a result, if we make a decision $x_t = X_t^{LA-SP,n}(S_t)$ and then step forward to $S_{t+1} = S^M(S_t, x_t, W_{t+1}(\omega))$, we have to solve (37) from scratch.

The need to recompute the lookahead model is not limited to deterministic approximations or approximations of stochastic models through scenario trees. We might solve the lookahead model using a value function, as we did previously in equation (31), repeated here for convenience:

$$X_t^{\pi}(S_t) = \arg\min_{x_{tt}\in\mathcal{X}_t} \big(C(S_{tt}, x_{tt}) + \mathbb{E}\{\tilde{V}_{t,t+1}(\tilde{S}_{t,t+1})|S_t\}\big). \tag{40}$$

Here, we might assume that $\tilde{V}_{t,t+1}(\tilde{S}_{t,t+1})$ is an exact estimate of the downstream value of being in state $\tilde{S}_{t,t+1}$ when solving our simplified lookahead model. Because the lookahead model typically involves information available at time t, when we make a decision at time t and step forward in time (in the real process), we generally have to start all over again from scratch. This is not the case when we use a value function approximation $\overline{V}_t(S_t)$ that is an approximation of the downstream value in the real model.

5.5 Evaluating a lookahead policy

The correct way to evaluate a lookahead policy (or any policy) is in the context of the objective function in (15). Figure 5 illustrates the process of solving a stochastic lookahead model using

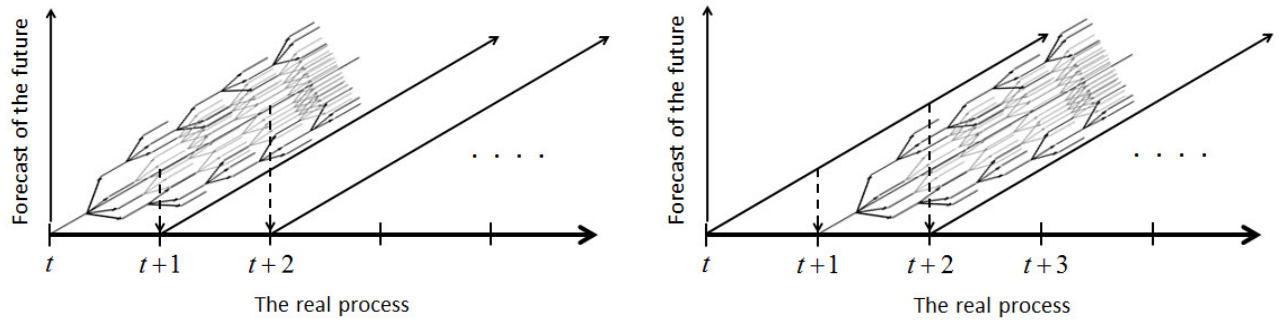


Figure 5: Illustration of rolling horizon procedure, using a stochastic model of the future (from Powell et al. (2012*b*)).

scenario trees, then stepping forward in time to simulate the performance of our lookahead policy. As of this writing, there is very little research evaluating the performance of lookahead policies. Considerably more attention has been given to this topic in the engineering controls community under the umbrella of model predictive control, but this work assumes a particular structure to the problem that generally does not apply in operations research (see Camacho & Bordons (2004) for a good introduction to MPC, and Bertsekas (2012) for a modern introduction to approximate dynamic programming and optimal control).

5.6 Comments

The distinction between models and lookahead models has not entered the literature, so care has to be used when deciding if a paper is solving the true model or just a lookahead model. It is our experience that the vast majority of papers using stochastic programming for multistage problems are using lookahead policies (see, for example, Jacobs et al. (1995), Takriti et al. (1996), Dupaçová et al. (2000), Wallace & Fleten (2003), Jin et al. (2011)). This means that after implementing the first-period decision and stepping forward in time, the problem has to be solved again with a new set of scenario trees.

But this is not always the case. Shapiro et al. (2013) formulates and solves a 120-period stochastic program (we would call it a dynamic program) using Benders cuts (a form of value function approximation) for an application that does not require the use of a scenario tree. While this could be viewed as a 120-period lookahead model, it is perfectly reasonable to define his original model as consisting of 120 periods (whether he needs 120 periods to make a good decision at time 0 is up for debate). Our own work (Powell et al. (2012*a*), Simao et al. (2009) and Powell & Topaloglu (2006) for example) also uses value functions that are approximations of the true model, over the entire horizon. In both sets of research, once the value functions (Benders cuts in Shapiro et al. (2013) and Sen & Zhou (2012)) have been estimated, they can be used not just to determine the first period decision, but all decisions over the horizon. These are examples of algorithms that are solving the true model rather than a lookahead model.

Scenario trees almost always imply a lookahead policy, because stepping forward in the real model will invariably put us in a state that is not represented in the scenario tree. Sen & Zhou (2012) introduces multi-stage stochastic decomposition (MSD) building on the breakthrough of stochastic decomposition for two-stage problems (Higle & Sen (1991)). The MSD algorithm generates scenarios which asymptotically cover the entire state space, but any practical implementation (which is limited to a finite number of samples) would still require re-optimizing after stepping forward in time, since it is unlikely that we would have sampled the state that we actually transitioned to.

Even with these simplifications, optimal solutions of the lookahead model may still be quite difficult, and a substantial literature has grown around the problem of solving stochastic lookahead models (Rockafellar & Wets (1991), Higle & Sen (1996), Romisch & Heitsch (2009), Birge & Louveaux (2011), King & Wallace (2012)). Often, exact solutions to even a simplified stochastic model are not achievable, so considerable attention has been given to estimating bounds. But it is important to realize:

- *An optimal solution to a lookahead model is, with rare exceptions, not an optimal policy.*
- *A bound on a lookahead model is not a bound on the performance of the resulting policy (with rare exceptions, such as Shapiro et al. (2013)).*

For an in-depth treatment of the properties of lookahead policies, see the work of Sethi (see Sethi & Haurie (1984), Sethi & Bhaskaran (1985) and Bhaskaran & Sethi (1987)). Not surprisingly, this literature is restricted to very simple problems.

This is a good time to return to the comment of the helpful referee who felt that multistage stochastic programming offered a richer framework than dynamic programming. The comment ignores the idea that the stochastic program is actually a lookahead policy for solving a dynamic program, and that what we care most about is the performance of the policy (that is, equation (33)) rather than the lookahead objective (equation (30)). This comment also ignores the fact that the lookahead model is itself a dynamic program (no matter how it is solved), and many in the stochastic programming community even use value function approximations (in the form of Benders cuts) to solve the lookahead model (see Shapiro et al. (2009) and Jacobs et al. (1995) for examples). We feel that placing stochastic programming within the broader framework of dynamic programming separates the lookahead model from the real model (which is usually ignored). In addition, it helps to build bridges to other communities (especially simulation), but raises new research questions, such as the performance of stochastic programming as a policy rather than the value of the lookahead objective function.

6 Direct policy search versus Bellman error minimization

Lookahead policies represent powerful approximations for many problems, and are especially popular for more complex problems. However, there are problems where it is possible to compute optimal

policies, and these can serve as useful benchmarks for approximate policies such as CFAs (estimated using direct policy search), and policies based on VFAs (estimated using Bellman error minimization).

Consider a policy given by

$$X_t^\pi(S_t|\theta) = \arg\min_{x\in\mathcal{X}_t}\left(C(S_t,x) + \gamma\sum_{f\in\mathcal{F}}\theta_f\phi_f(S_t^x)\right). \tag{41}$$

We now face a choice between two methods for choosing θ: Bellman-error minimization (which produces a value function approximation), or direct policy search (which gives us a cost function approximation). In this section, we are going to compare a series of strategies for estimating CFA- and VFA-based policies using a relatively simple energy storage application.

With direct policy search, we choose θ by solving equation (33) using any of a range of stochastic search methods (see Maxwell et al. (2010)[Section 5.4] for an example). If we use Bellman-error minimization, we need to create estimates of the value of being in a state S_t, and then use these estimates to update our value function approximation. One method, approximate value iteration, computes estimates $\hat{v}_t^n$ (in iteration n) of the value of starting in some state S_t^n using

$$\hat{v}_t^n = \min_{x\in\mathcal{X}_t}\big(C(S_t^n,x) + \overline{V}_t^{n-1}(S_t^{x,n})\big),$$

where $S_t^{x,n}$ is the post-decision state given that we are in state S_t^n and take action x.

A second method involves defining our policy using (41), and then simulating this policy from time t until the end of the horizon following sample path ω^n using

$$\hat{v}_t^n = \sum_{t'=t}^{T} C(S_{t'}^n, X^\pi(S_{t'}^n|\theta^{n-1})),$$

where $S_{t'+1} = S^M(S_{t'}^n, x_{t'}^n, W_{t'+1}(\omega^n))$. This is the foundation of approximate policy iteration, originally derived as backpropagation in the early work of Paul Werbos (Werbos (1974), Werbos (1989), Werbos (1992)) who introduced the idea of backward differentiation. It is beyond the scope of our presentation to discuss the merits of these two approaches (see chapters 9 and 10 of Powell (2011)), but the key idea is to create an observation (or set of observations) of the value of being in a state and then use these observations to obtain θ^n.

To provide a hint into the performance of these different methods, we have been running a series of experiments on a relatively simple set of energy storage problems (some infinite horizon, others finite horizon) which we are able to solve optimally, providing a benchmark. While this is just one class of applications, the results are consistent with our experiences on other problems. Here are the algorithms we have run:

1) A simple myopic policy (minimizes real-time costs now without regard to the future).

2) Basic least squares approximate policy iteration described in Lagoudakis & Parr (2003).

3) Least squares approximate policy iteration using instrumental variables (or projected Bellman error minimization), also described in Lagoudakis & Parr (2003) and implemented in Scott et al. (2013).

4) Direct policy search, using the same basis functions used in the testing with least squares policy iteration (Scott et al. (2013), using Scott et al. (2011) to do the stochastic search).

5) Approximate value iteration with a backward pass (known as TD(1) in the reinforcement learning community Sutton & Barto (1998) or backpropagation in the controls community Werbos (1992)), but using a piecewise linear value function (in the energy resource variable) which maintains concavity (see Salas & Powell (2013)).

6) Approximate policy iteration - Pham (2013) evaluated API using linear models, support vector regression, Gaussian process regression, tree regression, kernel regression, and a local parametric method, on a library of test problems with optimal solutions.

The first four problems were run on a steady state (infinite horizon) problem, while the rest were all run on a finite horizon, time-dependent set of problems. Optimal policies were found using classical value iteration to within 1 percent of optimality (with a discount factor of .9999, given the very small time steps). Optimal policies were calculated by discretizing the problems and solving them using classical (model-based) backward dynamic programming. These datasets were also evaluated by Pham (2013) who tested a range of learning algorithms using approximate policy iteration.

Figure 6 shows the results of the first four experiments, all scaled as a percent of the optimal. The runs using direct policy search were almost all over 90 percent of optimality, while the runs using least squares policy iteration, which is based on Bellman error minimization, were much worse. By contrast, the runs using approximate value iteration but exploiting concavity all consistently produced results over 99 percent of optimality on the finite horizon problems (Salas & Powell (2013)).

These observations need to be accompanied by a few caveats. The use of the piecewise linear value function approximation (used in Salas & Powell (2013)) scales well to time-dependent problems with hundreds or thousands of storage devices and hundreds to thousands of time periods, but cannot handle more than one or two "state of the world" variables which might capture information about weather, prices and demands. Direct policy search works well when searching over a small number of parameters (say, two or three), but would not scale to a time dependent problem where the parameters vary over time (the VFA-based approach had no trouble with many time periods).

From this work (and related computational experience), we have been drawing the following conclusions:

- Bellman error minimization works extremely well when we exploited convexity (Salas & Powell (2013)), but surprisingly poorly (e.g. around 70 percent of optimal) using simple, quadratic

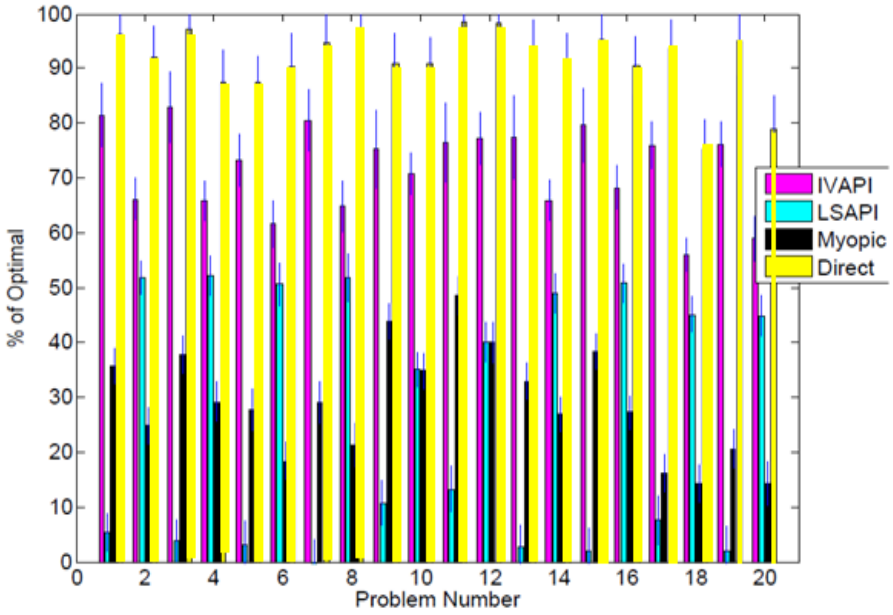


Figure 6: The performance of a series of approximation algorithms relative to the optimal solution for 20 benchmark storage problems. These include: a myopic policy, a basic form of least squares approximate policy iteration, LSAPI with instrumental variables, and direct policy search using the same structure for the policy (from Scott et al. (2013)).

basis functions (which appeared to provide very reasonable estimates of the value function). Higher order basis functions performed even worse.

- Policy search using the same structure of policy (same basis functions) performed very well (around 95 percent of optimal). However, this is limited to low dimensional parameter vectors, which excludes problems where the regression vector θ is a function of time.

- Approximate value iteration when exploiting convexity works best of all (over 98 percent of optimal), but depends on our ability to approximate the value of multiple storage devices independently using low-dimensional lookup tables where convexity is maintained (in the controllable dimension). Without convexity, we have had no success with either approximate value iteration or its close cousin, Q-learning (see Sutton & Barto (1998)).

- A deterministic lookahead policy underperformed approximate value iteration, but performed as well as policy search (approximately 95-99 percent of optimal). The lookahead policy did not require any training, but would require more computation compared to using a value function once the VFA has been fitted.

- Approximate policy iteration, which seems to enjoy much better convergence theory (see e.g. Bertsekas, 2012), worked adequately. Our best results on the storage problems, using support vector regression, might get over 80 percent of optimal (sometimes to 90 percent) after

10 policy iterations (each iteration required several thousand policy simulations). However, support vector regression is not well suited to recursive estimation, needed in algorithms such as approximate value iteration.

One conclusion that emerged consistently from these results was that the presence of an optimal benchmark provided tremendous insights into the quality of the solution. It was only through these benchmarks that we were able to see that many of these apparently sophisticated algorithms actually work quite poorly, even on a relatively simple problem class such as energy storage (we would never have been able to get optimal benchmarks on more complex problems). We have been able to get near-optimal solutions using value iteration, but only when we could exploit structure such as convexity or concavity, and only when using a lookup table representation, which clearly limits the scalability of these approaches to additional state variables. We were never able to get high quality solutions using linear models using approximate value iteration (this was not a surprise, given known theory) or approximate policy iteration (this was more of a surprise).

We note in closing this discussion that a number of authors have recognized that you can "tune the value function" for a policy such as (28) using direct policy search (see Maxwell et al. (2013) for a good example of this). However, if you are choosing θ to tune the policy in (28), we would argue that the regression term $\sum_{f\in\mathcal{F}}\theta_f\phi_f(S^x)$ is no longer an approximate value function. For example, if your decision variable is x, there is no reason to include any basis function that does not depend on x if you are doing policy search, where all we care about is the optimal x given θ. For this reason, if we are doing policy search, there is no reason to expect $\sum_{f\in\mathcal{F}}\theta_f\phi_f(S^x)$ to bear any relationship to a value function.

If you are using policy search, we would argue that you are using a cost function approximation, where the CFA policy is given by

$$\begin{aligned} X^{CFA}(S_t|\theta) &= \arg\min_{x_t} \bar{C}^{\pi}(S_t,x_t|\theta) \\ &= \arg\min_{x_t}\left(C(S_t,x_t)+\sum_{f\in\mathcal{F}}\theta_f\phi_f(S_t^x)\right). \end{aligned} \tag{42}$$

When we do direct policy search, there is no reason to include any basis functions that are not a function of x_t, since terms that do not depend on x_t are simply constants in the argmin in equation (42). In our energy storage example, we used basis functions such as S_tx_t, x_t and x_t^2. Thus, there is no reason to expect the correction term (involving the basis functions) to approximate in any way the downstream value of being in state S_t^x.

We suspect that such an approach for designing a cost function approximation probably works well when the myopic policy

$$X^{\pi}(S_t) = \arg\min_{x_t\in\mathcal{X}_t} C(S_t,x_t)$$

works "reasonably well." In our energy storage experiments, the myopic policy produced results that were about 30 percent of optimal, which is hardly stellar, but it is not hard to think of examples where a myopic policy would be terrible.

7 How do we choose a policy?

Instead of making choices such as "stochastic programming," "dynamic programming," or "simulation," we would like to ask readers to pose their choice as between "lookahead policies," "policies based on value function approximations," or "policy function approximations." Deterministic lookahead policies which have been tweaked to produce more robust solutions (such as introducing safety stocks or reserve capacities) should be viewed as lookahead cost function approximations. Cost function approximations are widely used in practice, but are generally dismissed as some sort of heuristic compared to more elegant and "sophisticated" policies based on stochastic lookaheads ("stochastic programming") or value function approximations ("approximate dynamic programming"). We would argue that all of these methods are, in the end, approximations that have to be tested, and there is no reason to believe a priori that one will be better than another on a specific problem.

The reality is that with rare exceptions, all of these classes of policies are almost guaranteed to be suboptimal. There is no apriori reason why a cost function approximation, policy function approximation, policy based on value function approximation, or a lookahead which provides an optimal solution to an approximate model, should be better than all the rest. Each class of policy offers specific features that could produce superior results for a specific problem class.

So how to choose? Based on our experience, the following comments might provide guidance:

- Policy function approximations work best for low-dimensional actions, where the structure of the policy is fairly obvious. (s, S) inventories are an easy example. Another is that we might want to sell a stock when its price goes above a particular price. Policy function approximations can also work well when the policy is a relatively smooth surface, allowing it to be approximated perhaps by a linear function (known in the literature as "affine policies") or locally linear functions.

- Cost function approximations, which are typically variations of deterministic models, work best when a deterministic model works well, and when the impact of uncertainty is easy to recognize. It may be easy to see, for example, that we should provide a buffer stock to protect against supply chain disruptions. Cost function approximations can be particularly attractive when the decision x_t is multidimensional, since we can typically solve a CFA-based policy using a solver.

- Value function approximations are particularly useful when the value of the future given a state is easy to approximate. When solving multidimensional resource allocation problems, keep in mind that the value function only needs to communicate the marginal value, not the value of

being in a state. Also, the issue of value functions has nothing to do with size (we have used value function approximations on very high dimensional applications in transportation with state variables with 10,000 dimensions or more). Large problems can be easily approximated using separable approximations. The real issue is nonseparable interactions. These are easy to capture in a lookahead model, but are hard to approximate using a statistical model.

- Lookahead policies are particularly useful for time-dependent problems, and especially if there is a forecast available that evolves over time. The parameters of a stochastic lookahead model (e.g. the number of scenarios in a scenario tree, the number of stages, the horizon) should always be tested in a simulator that is able to compare the performance of different policies. Also, a stochastic lookahead model should always be compared to the performance of a deterministic lookahead model. Just because the underlying problem is stochastic does not mean that a deterministic lookahead model will not work.

Hybrid policies can be particularly valuable. Instead of building a lookahead policy over a long horizon H, we can use a shorter horizon but then introduce a simple value function approximation at the end. Cost function approximations over planning horizons can make a deterministic approximation more robust. Finally, you can tune a high-dimensional myopic policy (e.g. assigning drivers to loads or machines to tasks) with low-order policy function approximations ("assign team drivers to long loads") by adding the low order policy functions as bonus or penalty terms to the objective function.

We hope this discussion has helped to place stochastic programming, dynamic programming, and simulation (using policy function approximations) in a common framework. Over time, these terms have evolved close associations with specific classes of policies (lookahead policies, value functions, and policy function approximations, respectively). It is for this reason that we suggest a new name, *computational stochastic optimization*, as an umbrella for all of these fields (and more).

Acknowledgements

This research was supported by the National Science Foundation grant CMMI-0856153.

References

Andradóttir, S. & Prudius, A. A. (2010), 'Adaptive Random Search for Continuous Simulation Optimization', *Naval Research Logistics (NRL)* **57**(6), 583–604.

Bandi, C. & Bertsimas, D. J. (2012), 'Tractable stochastic analysis in high dimensions via robust optimization', *Mathematical Programming Series B* **134**, 23–70.

Barr, D. R. & Rizvi, M. H. (1966), 'An introduction to ranking and selection procedures', *J. Amer. Statist. Assoc.* **61**(315), 640– 646.

Bellman, R. E. (1957), 'Dynamic Programming', *Princeton University Press, Princeton, NJ.*

Ben-Tal, A., Ghaoui, L. E. & Nemirovski, A. (2009), *Robust Optimization*, Princeton University Press, Princeton NJ.

Bertsekas, D. P. (2012), *Dynamic Programming and Optimal Control, Vol. II, 4th Edition: Approximate Dynamic Programming*, Athena Scientific, Belmont, MA.

Beyer, H. & Sendhoff, B. (2007), 'Robust optimization A comprehensive survey', *Computer Methods in Applied Mechanics and Engineering* **196**(33-34), 3190–3218.

Bhaskaran, S. & Sethi, S. P. (1987), 'Decision and Forecast Horizons in a Stochastic Environment : A Survey', *Optimal Control Applications and Methods* **8**, 61–67.

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

Boesel, J., Nelson, B. L. & Kim, S. (2003), 'Using ranking and selection to "clean up" after simulation optimization', *Operations Research* **51**(5), 814–825.

Bouzaiene-Ayari, B., Cheng, C., Das, S., Fiorillo, R. & Powell, W. B. (2012), From Single Commodity to Multiattribute Models for Locomotive Optimization : A Comparison of Integer Programming and Approximate Dynamic Programming.

Camacho, E. F. & Bordons, C. (2004), *Model Predictive Control*, Springer Verlag, New York.

Chang, H. S., Fu, M. C., Hu, J. & Marcus, S. I. (2007), *Simulation-based Algorithms for Markov Decision Processes*, Springer, Berlin.

Chick, S. E. & Gans, N. (2009), 'Economic analysis of simulation selection problems', *Management Science* **55**(3), 421–437.

Chick, S. E., He, D. & Chen, C.-h. (2007), 'Opportunity Cost and OCBA Selection Procedures in Ordinal Optimization for a Fixed Number of Alternative Systems', *IEEE Transactions on Systems Man and Cybernetics Part C-Applications and Reviews* **37**(5), 951–961.

Defourny, B., Ernst, D. & Wehenkel, L. (2013), 'Scenario Trees and Policy Selection for Multistage Stochastic Programming using Machine Learning', *Informs J. on Computing* pp. 1–27.

Dorfman, R. (1984), 'The Discovery of Linear Programming', *Annals of the History of Computing* **6**(3), 283–295.

Dupaçová, J., Consigli, G. & Wallace, S. (2000), 'Scenarios for multistage stochastic programs', *Annals of Operations Research* **100**, 25–53.

Fu, M. C., Glover, F. & April, J. (2005), 'Simulation optimization: a review, new developments, and applications', *Proceedings of the 37th conference on Winter simulation* pp. 83—-95.

Gittins, J., Glazebrook, K. & Weber, R. R. (2011), *Multi-Armed Bandit Allocation Indices*, John Wiley & Sons, New York.

Hastie, T., Tibshirani, R. & Friedman, J. (2009), *The elements of statistical learning: data mining, inference and prediction*, Springer, New York.

Higle, J. & Sen, S. (1991), 'Stochastic decomposition: An algorithm for two-stage linear programs with recourse', *Mathematics of Operations Research* **16**(3), 650–669.

Higle, J. & Sen, S. (1996), *Stochastic Decomposition: A Statistical Method for Large Scale Stochastic Linear Programming*, Kluwer Academic Publishers.

Hong, L. & Nelson, B. L. (2007), 'A framework for locally convergent random-search algorithms for discrete optimization via simulation', *ACM Transactions on Modeling and Computer Simulation* **17**(4), 1–22.

Jacobs, J., Freeman, G., Grygier, J., Morton, D., Schultz, G., Staschus, K. & Stedinger, J. (1995), 'SOCRATES-A system for scheduling hydroelectric generation under uncertainty', *Ann. Oper. Res* **59**, 99–133.

Jin, S., Ryan, S., Watson, J. & Woodruff, D. (2011), 'Modeling and solving a large-scale generation expansion planning problem under uncertainty', *Energy Systems* **2**, 209–242.

Kantorovich, L. (1939), 'Matematicheskie Metody Organizatsii i Planirovaniya', *Leningrade State University Press*.

King, A. J. & Wallace, S. (2012), *Modeling with stochastic programming*, Springer Verlag, New York.

Kleywegt, A. J., Shapiro, A. & Homem-de Mello, T. (2002), 'The sample average approximation method for stochastic discrete optimization', *SIAM J. Optimization* **12**(2), 479–502.

Koopmans, T. (1947), 'Optimum utilization of the transportation system', *Econometrica* **17**, 136–146.

Lagoudakis, M. & Parr, R. (2003), 'Least-squares policy iteration', *Journal of Machine Learning Research* **4**, 1107–1149.

Maxwell, B. M. S., Henderson, S. G. & Topaloglu, H. (2013), 'Tuning approximate dynamic programming policies for ambulance redeployment via direct search', *Stochastic Systems* **3**(2), 322–361.

Maxwell, M. S., Restrepo, M., Henderson, S. G. & Topaloglu, H. (2010), 'Approximate Dynamic Programming for Ambulance Redeployment', *INFORMS Journal on Computing* **22**(2), 266–281.

Pereira, M. F. & Pinto, L. M. V. G. (1991), 'Multi-stage stochastic optimization applied to energy planning', *Mathematical Programming* **52**, 359–375.

Pham, T. (2013), Experiments with Approximate Policy Iteration, PhD thesis, Princeton University.

Powell, W. B. (2007), *Approximate Dynamic Programming: Solving the curses of dimensionality*, John Wiley & Sons, Hoboken, NJ.

Powell, W. B. (2011), *Approximate Dynamic Programming: Solving the curses of dimensionality*, 2 edn, John Wiley & Sons, Hoboken, NJ.

Powell, W. B. & Ryzhov, I. O. (2012), *Optimal Learning*, John Wiley & Sons, Hoboken, NJ.

Powell, W. B. & Topaloglu, H. (2006), 'Dynamic-programming approximations for stochastic time-staged integer multicommodity-flow problems', *Informs Journal on Computing* **18**(1), 31.

Powell, W. B., George, A., Simão, H. P., Scott, W. R., Lamont, A. D. & Stewart, J. (2012*a*), 'SMART : A Stochastic Multiscale Model for the Analysis of Energy Resources , Technology , and Policy', *Informs J. on Computing* **24**(4), 665–682.

Powell, W. B., Simao, H. P. & Bouzaiene-Ayari, B. (2012*b*), 'Approximate dynamic programming in transportation and logistics: a unified framework', *EURO Journal on Transportation and Logistics* **1**(3), 237–284.

Puterman, M. L. (2005), *Markov Decision Processes*, Vol. 7pp, 2nd edn, John Wiley and Sons, Hoboken, NJ.

Rockafellar, R. T. & Wets, R. J.-B. (1991), 'Scenarios and policy aggregation in optimization under uncertainty', *Mathematics of Operations Research* **16**(1), 119–147.

Romisch, W. & Heitsch, H. (2009), 'Scenario tree modeling for multistage stochastic programs', *Mathematical Programming* **118**, 371–406.

Ruszczynski, A. (2010), 'Risk-averse dynamic programming for Markov decision processes', *Mathematical Programming* **125**(2), 235–261.

Salas, D. F. & Powell, W. B. (2013), Benchmarking a Scalable Approximate Dynamic Programming Algorithm for Stochastic Control of Multidimensional Energy Storage Problems.

Scott, W. R., Frazier, P. & Powell, W. B. (2011), 'The Correlated Knowledge Gradient for Simulation Optimization of Continuous Parameters using Gaussian Process Regression', *SIAM Journal on Optimization* **21**(3), 996.

Scott, W. R., Powell, W. B. & Moazeni, S. (2013), Least Squares Policy Iteration with Instrumental Variables vs . Direct Policy Search : Comparison Against Optimal Benchmarks Using Energy Storage.

Sen, S. & Zhou, Z. (2012), 'Multistage Stochastic Decomposition : A Bridge between Stochastic Programming and Approximate Dynamic Programming 1 Introduction', *SIAM J. Optimization* pp. 0–19.

Sethi, S. P. & Bhaskaran, S. (1985), 'Conditions for the Existence of Decision Horizons for Discounted Problems in a Stochastic Environment', *Operations Research Letters* **4**, 61–64.

Sethi, S. P. & Haurie, A. (1984), 'Decision and Forecast Horizons, Agreeable Plans, and the Maximum Principle for Infinite Horizon Control Problem', *Operations Research–letters* **3**, 261–265.

Shapiro, A., Dentcheva, D. & Ruszczynski, A. (2009), *Lectures on stochastic programming: modeling and theory*, SIAM, Philadelphia.

Shapiro, A., Tekaya, W., Paulo, J. & Pereira, M. F. (2013), 'Risk neutral and risk averse Stochastic Dual Dynamic Programming method', *European J. Operational Research* **224**, 375–391.

Simao, H. P., Day, J., George, A., Gifford, T., Powell, W. B. & Nienow, J. (2009), 'An Approximate Dynamic Programming Algorithm for Large-Scale Fleet Management: A Case Application', *Transportation Science* **43**(2), 178–197.

Spall, J. C. (2003), *Introduction to Stochastic Search and Optimization: Estimation, Simulation and Control*, John Wiley & Sons, Hoboken, NJ.

Sutton, R. S. & Barto, A. G. (1998), *Reinforcement Learning*, Vol. 35, MIT Press, Cambridge, MA.

Swisher, J. R., Hyden, P. D., Jacobson, S. H. & Schruben, L. W. (2000), 'A survey of simulation optimization techniques and procedures', *Proceedings of the winter simulation conference* pp. 119–128.

Takriti, S., Birge, J. R. & Long, E. (1996), 'A stochastic model for the unit commitment problem', *Power Systems, IEEE Transactions on* **11**(3), 1497–1508.

Wallace, S. & Fleten, S.-E. (2003), Stochastic Programming Models in Energy, *in* A. Ruszczynski & A. Shapiro, eds, 'Stochastic Programming', Vol. 10, Elsevier Science B.V., Amsterdam, chapter 10, pp. 637–677.

Werbos, P. J. (1974), Beyond regression: new tools for prediction and analysis in the behavioral sciences, PhD thesis, Harvard University.

Werbos, P. J. (1989), 'Backpropagation and neurocontrol: A review and prospectus', *Neural Networks* pp. 209—-216.

Werbos, P. J. (1992), Approximate Dynamic Programming for Real-Time Control and Neural Modelling, *in* D. J. White & D. A. Sofge, eds, 'Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches'.