TUTORIALS IN
OPERATIONS RESEARCH
INFORMS 2014

 | ISBN 978-0-9843378-5-9
http://dx.doi.org/10.1287/educ.2014.0128

Clearing the Jungle of Stochastic Optimization

Warren B. Powell
Department of Operations Research and Financial Engineering, Princeton University, Princeton, New Jersey 08544, powell@princeton.edu

Abstract Whereas 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 versus optimization, stochastic programming versus dynamic programming) can be reduced to four fundamental classes of policies that are evaluated in a simulation-based setting we call the base model. 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.

Keywords stochastic optimization; stochastic programming; approximate dynamic programming; reinforcement learning

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 (see, e.g., Vanderbei [66])

$$\min_x \quad cx \tag{1}$$

$$\text{subject to} \quad Ax = b, \tag{2}$$

$$x \geq 0. \tag{3}$$

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

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

$$\text{subject to} \quad A_0 x_0 = b_0, \tag{5}$$

$$A_t x_t - B_{t-1} x_{t-1} = b_t, \quad t = 1,\ldots,T, \tag{6}$$

$$x_t \geq 0, \quad t = 0,\ldots,T. \tag{7}$$

First introduced by Kantorovich in 1939 and then by Koopmans, linear programming was made famous by Dantzig with the introduction of the simplex method (a nice review of the history is given in Dorfman [18]), which transformed Equations (1)–(3) (or (4)–(7)) from a mathematical statement to a powerful tool for solving a wide range of problems. Although 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 that 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, and 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. Although these communities generally coexist fairly peacefully, there are ongoing debates between the use of simulation versus optimization, or stochastic programming versus dynamic programming. A helpful referee for a recent paper (Powell et al. [44]) offered the following 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 (Simão et al. [59]) and planning locomotives at one of the largest railroads in the country (Bouzaiene-Ayari et al. [13]). 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. [45]). How were these ultra high-dimensional stochastic optimization problems solved? They were solved by 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:

- A dynamic program 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 classes of dynamic programs.
- A stochastic program (for multistage problems) is, with some exceptions, a look-ahead policy that involves solving a look-ahead model (which is itself a dynamic program) for solving a larger dynamic program. See §5.6 for a discussion of an exception.

• 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—see §5.6).

• 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, it is quite rare to actually find an optimal policy.

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 (or more precisely, policies). We follow this discussion with a presentation of what we feel is the correct way to model a sequential decision process (that is, a dynamic program), 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:

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

where

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

This is the format introduced by Bellman [6] and popularized by many authors (such as Puterman [46], the last of a long line of books on Markov decision processes). The format is elegant and has enabled a rich theoretical tradition. A well-known concept in dynamic

programming is that of a *policy* that 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

$$\pi(s) = \arg\min_{a \in \mathcal{A}} \left(C(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V_{t+1}(s') \right). \tag{9}$$

Here, we are using the standard notation $\pi(s)$ for policy, where Equation (9) would produce the optimal policy (if we could compute it). The most difficult challenge with Markov decision processes is that researchers often assume that states are discrete and that value functions and policies use lookup table representations. This sharply limits its applicability to relatively small problems. Below, we modify this notation so that we use "π" in a more practical way.

2.2. A Stochastic Programming "Model"

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

$$\min_{x_t, (x_{t'}(\omega), t < t' \le t+H), \forall \omega \in \Omega_t} \left(c_t x_t + \sum_{\omega \in \Omega_t} p(\omega) \sum_{t'=t+1}^{t+H} c_{t'}(\omega) x_{t'}(\omega) \right). \tag{10}$$

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 as

$$A_t x_t = b_t, \tag{11}$$

$$x_t \ge 0. \tag{12}$$

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 as

$$A_{t'}(\omega) x_{t'}(\omega) - B_{t'-1}(\omega) x_{t'-1}(\omega) = b_{t'}(\omega) \quad \forall \omega \in \Omega_t, \tag{13}$$

$$x_{t'}(\omega) \ge 0 \quad \forall \omega \in \Omega_t. \tag{14}$$

Note that we require that $x_t = x_t(\omega)$, a condition that is often imposed as a *nonanticipativity constraint*.

Although 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 (10)–(14) is known as a two-stage approximation; it is widely used simply because the multistage version is completely intractable for most applications (even when we use Monte Carlo sampling). 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 (10)–(14)) is not actually a model; they are actually classes of policies for solving dynamic programs. 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). This sequential decision problem is, in fact, a dynamic program, which is a problem that we want to solve by finding the best policy.

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

• *State* S_t. This represents what we know (more precisely, what we need to 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). Later, we defer to the choice of how a decision is made, but we 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 $A_t^\pi(S_t)$ (or $A^\pi(S_t)$ if our policy is stationary). 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 [42, Chap. 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

$$h_t = (W_1, \ldots, W_t). \tag{15}$$

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 [46]), or the sequence comprising the 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 (15).

• *The transition function.* We write this as $S_{t+1} = S^M(S_t, x_t, W_{t+1})$, where $S^M(\cdot)$ has been referred to as the system model, plant model, transfer function, or just model, but we refer to it as the transition function, consisting of the equations that describe the evolution of the system from t to $t+1$. 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 could write

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

or, if we want $\hat{p}_{t+1}$ to be the relative change, we might write

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

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

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

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 researchers using different approaches (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 nonadditivity (see Ruszczyński [49] and Shapiro et al. [57] and for thorough discussions). For example, we might replace (16) with

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

where $Q_{\alpha}(W)$ is the α-quantile of the random variable W (feel free to substitute your favorite risk measure here; see Rockafellar and Uryasev [47]). Alternatively, we might use a *robust* objective that uses

$$\min_{\pi \in \Pi} \max_{W \in \mathcal{W}} \sum_{t=0}^{T} C(S_t, X_t^{\pi}(S_t)), \tag{18}$$

where the maximum over $W \in \mathcal{W}$ refers to a search over random variables within an uncertainty set $\mathcal{W}$ (Bandi and Bertsimas [2]) that defines a set of "worst-case" values. We might replace the set $W \in \mathcal{W}$ with $\omega \in \Omega$ to be consistent with our notation. See Beyer and Sendhoff [9] and Ben-Tal et al. [7] 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 (16), should be compared to its deterministic counterpart, Equation (4). The linking Equation (6) can be written as

$$A_t x_t = R_t, \tag{19}$$

$$R_t = b_t + B_{t-1} x_{t-1}. \tag{20}$$

If we let $S_t = R_t$, we see that Equation (6), in the form of Equations (19) and (20), 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 representation 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)). In stochastic optimization, we are looking for the best policy (see (16)), 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.

We would like to refer to our representation of states, actions, exogenous information, transition function, and objective function as "the model." However, in the discussion below we are going to introduce the concept of the look-ahead model, which is also widely referred to in the literature as "the model." For this reason, we need to make a distinction between the problem we are trying to solve (the model) and any approximations that we may use for the sole purpose of making a decision ("the policy"), which often involves solving an approximate model.

For this reason, we suggest referring to our representation of the problem as the *base model*, and we propose to use the objective function (16) (or (17) or (18)) as the expression that represents our base model, since all the other elements of the model are implied by this single equation. The base model, of course, represents an approximation of some real ("true") problem, but these approximations are the purview of the modeler. Our job, as algorithms specialists, is to do the best we can to solve the base model, producing the highest-quality policy that solves (16) (or (17) or (18)). It is not our job to question simplifications made in the formulation of the base model, but it is our responsibility to challenge approximations that might affect the performance of policies when evaluated in terms of the objective function for the base model (whatever we choose that to be).

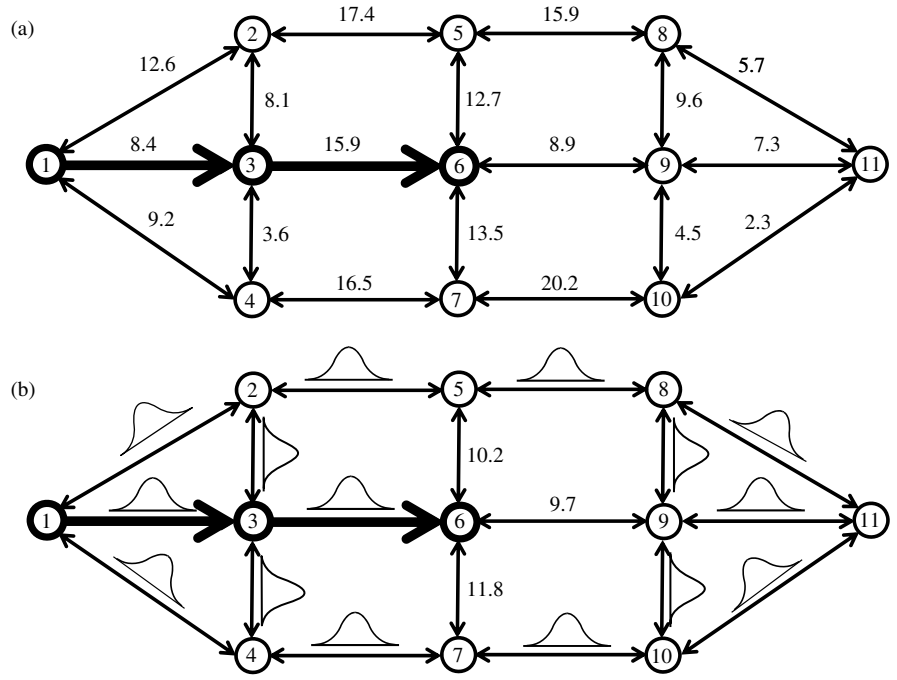
This style of modeling problems has not been used in operations research (or computer science) but is widely used in the engineering controls community (but even popular control theory texts such as Lewis et al. [33] do not articulate the elements of a model as we have). The Markov decision process community prefers to assume that the one-step transition matrix (used in Equation (8)) is given (see Puterman [46], for example). Although 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).

The two dimensions of our modeling 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 [6, p. 81] offers that "...we have a physical system characterized at any

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.



stage by a small set of parameters, the *state variables*." Puterman [46] introduces state variables as follows: "At each decision epoch, the system occupies a *state*." (Italics are in the original manuscripts.) *Wikipedia*[1] (in 2013) suggests, "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 panel (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

$$S_t = N_t = 6.$$

Note that we exclude from our state variable all data that are not changing (the arc costs are all assumed to be deterministic). Some will insist that the state variable should include the history (that is, nodes 1 and 3), but for this model, this information is not relevant.

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, panel (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. Although 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 (the costs out of the node).

[1] *Wikipedia*. Accessed July 20, 2014, http://en.wikipedia.org/wiki/State_Variable.

For our final example, we introduce the twist that there are left-hand turn penalties for our stochastic network in Figure 1, panel (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 [42], I offer the following definition.

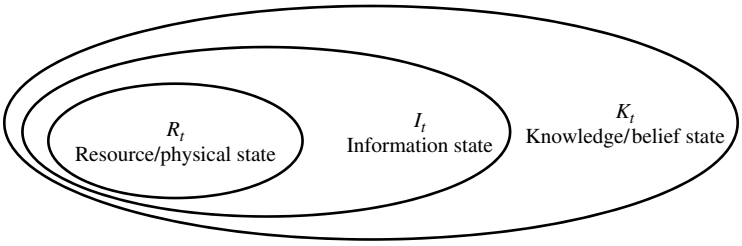
Definition 3.1. A *state variable* is the minimally dimensioned function of history that is necessary and sufficient to compute the decision function, the transition function, and the contribution function.

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

- *Physical (or resource state) R_t.* We typically use "resource state" to describe the status of people, equipment, products, money, and energy (how much and where). We might use the term "physical state" to describe the attributes of a physical resource (e.g., the location and velocity of an aircraft, the position of the arms of a robot, the temperature of a chemical).
- *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, multiarmed bandit problems (Gittins et al. [21]), and optimal learning (Powell and Ryzhov [43]). We only include these distributions in our state variable if they are changing as we collect more information (hence the reason we excluded them in our network example above).

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, whether a piece of data is classified as part of the physical/resource state or the information state is not relevant. We make the distinction only because it seems to help people think about a problem. For example, many people fall in the trap of equating "state" with "physical state." We find it is useful to make a distinction between data (which we list under physical state or information state) and probability distributions, which we categorize under knowledge state.

FIGURE 2. Illustration of the growing sets of state variables, where the 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 as

$$F_t^\pi(S_t) = \mathbb{E}\left\{\sum_{t'=t}^{T} C(S_{t'}, X_{t'}^\pi(S_{t'})) \,\middle|\, S_t\right\}. \tag{21}$$

(Note to probabilists: throughout our presentation, we use the convention that the conditional expectation $\mathbb{E}\{\cdot \mid S_t\}$ means the expectation over the probability space $(\Omega_t, \mathcal{F}_t, \mathcal{P}_t)$, where Ω_t is the set of outcomes constructed given we are in a known state S_t (which is not random at time t), and where $\mathcal{F}_t$ is the sigma-algebra on Ω_t, and $\mathcal{P}_t$ is the conditional probability measure given that we are in state S_t.)

We can, of course, write this as

$$F_t^\pi(S_t) = C(S_t, X_t^\pi(S_t)) + \mathbb{E}\left\{\sum_{t'=t+1}^{T} C(S_{t'}, X_{t'}^\pi(S_{t'})) \,\middle|\, S_t\right\} \tag{22}$$

$$= C(S_t, X_t^\pi(S_t)) + \mathbb{E}\{F_{t+1}^\pi(S_{t+1}) \mid S_t\}. \tag{23}$$

Now it is apparent that S_t (or in the case of (23), 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 $t' = t, \ldots, T$. Not surprisingly, we see no need to include any information in S_t that is not needed to compute any of these functions.

Although 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). (Systems with unobservable states are more subtle—our experience is that these can be handled by using probability distributions to represent what we know about an unobservable state.) 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!"

Finally, there is a surprisingly widespread tendency to assume that state variables have to be discrete, and if they are multidimensional, the curse of dimensionality kicks in, which means they are not useful. We note that (1) state variables do not have to be discrete, (2) it is not necessary to use lookup representations for functions, and most important (3) it is important to separate the process of modeling a problem from the design of a strategy for identifying an effective policy. Model your problem first with a generic policy $X_t^\pi(S_t)$, and then go looking for a policy (as we do below).

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 Kantorovich's statement of a linear program, and the reason why linear programming became so exciting after Dantzig introduced the simplex algorithm. Dantzig made Kantorovich'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 look-ahead 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

$$X^\pi(S_t \mid \theta) = \theta_0 + \theta_1 S_t + \theta_2 S_t^2. \tag{24}$$

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 job shop) that are governed by simple rules (we would call these policy function approximations).

• *Optimizing a cost function approximation* (*CFA*). There are problems where a simple myopic policy can produce good (in rare cases optimal) results. A myopic policy would be written

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

Not surprisingly, this would rarely work well in practice. However, there are problems where a slightly modified cost function might work quite well. One class of approximations looks like

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

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) that might be of the form $S_t x$, $S_t x^2$, x, x^2, which serves as a type of correction term. However, there are other problems where we make direct changes to the cost function itself, or perhaps the constraints (e.g., imposing a minimum inventory level). We can represent this class of policies more broadly by writing

$$X_t^\pi(S_t \mid \theta) = \arg\min_{x \in \mathcal{X}_t} \bar{C}^\pi(S_t, x \mid \theta), \tag{26}$$

where $\bar{C}_t^\pi(S_t, x \mid \theta)$ is some sort of parametric approximation. Here, we would let π carry the information about the structure of the approximation (such as the basis functions in Equation (25)), and we let θ capture all tunable parameters.

We have written the CFA in terms of costs over a single time period, but as we point out below, it is very common to use hybrids, and we may combine the concept of a CFA with a look-ahead policy.

- *Policies that depend on a value function approximation* (*VFA*). These are the policies most often associated with dynamic programming and are written as

$$X_t^\pi(S_t \mid \theta) = \arg\min_{x \in \mathcal{X}_t} \big(C(S_t, x) + \mathbb{E}\{ \bar{V}_{t+1}(S_{t+1} \mid \theta) \mid S_t \} \big), \tag{27}$$

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

$$X_t^\pi(S_t \mid \theta) = \arg\min_{x \in \mathcal{X}_t} \big(C(S_t, x) + \bar{V}_t(S_t^x \mid \theta) \big). \tag{28}$$

In both (27) and (28), 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

$$X_t^\pi(S_t \mid \theta) = \arg\min_{x \in \mathcal{X}_t} \left(C(S_t, x) + \sum_{f \in \mathcal{F}} \theta_f \phi_f(S_t^x) \right). \tag{29}$$

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 although (25) and (29) look similar, the mechanisms for fitting the regression coefficients are completely different, and it is unlikely that we would use the same basis functions. In (29), 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 (note that in (25), we would not include any basis functions that are not a function of x).

• *Look-ahead policies.* The simplest look-ahead 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 look-ahead model that has been triggered at time t. Variables with tildes represent the look-ahead model, so we do not confuse them with the base model. A deterministic look-ahead policy might be written

$$X_t^\pi(S_t \mid \theta) = \arg\min_{\tilde{x}_{tt},\ldots,\tilde{x}_{t,t+H}} \sum_{t'=t}^{t+H} C(\tilde{S}_{tt'}, \tilde{x}_{tt'}). \tag{30}$$

Normally, we optimize over the horizon $(t, \ldots, t+H)$ but only implement the first decision, so $X_t^\pi(S_t \mid \theta) = \tilde{x}_{tt}$. All variables in the look-ahead model are indexed by (tt'), where t represents when the decision is being made (and therefore its information content), and t' is when the decision impacts the physical system. The look-ahead variables (with tildes) may capture various approximations; for example, our base model may step forward in five-minute increments, whereas the look-ahead model may uses hourly increments so it is easier to solve. Here, the parameter θ captures all the parameters that determine the formulation of the look-ahead model (including choices such as the planning horizon).

We might also use a stochastic look-ahead model

$$\min_{\tilde{x}_{tt},\,(\tilde{x}_{tt'}(\tilde{\omega}),\,t<t'\le t+H),\,\forall\,\tilde{\omega}\in\tilde{\Omega}_t} \left(\tilde{c}_{tt}\tilde{x}_{tt} + \sum_{\tilde{\omega}\in\tilde{\Omega}_t} p(\tilde{\omega}) \sum_{t'=t+1}^{t+H} \tilde{c}_{tt'}(\tilde{\omega})\tilde{x}_{tt'}(\tilde{\omega}) \right). \tag{31}$$

In this case, θ captures parameters such as the number of information stages and the number of samples per stage (this community refers to the elements of $\tilde{\Omega}_t$ as *scenarios*). An "information stage" consists of revealing information, followed by making decisions that use this information. We need to construct a look-ahead stochastic process, captured by the set $\tilde{\Omega}_t$, to differentiate stochastic *scenarios* in the look-ahead model from the sample path $\omega \in \Omega$ that we might be following in the base model (we suggest using "sample path" to describe the evolution of information in the base model, and "scenario" to represent the evolution of information in the look-ahead model). The choice of the number of stages and the construction of the set $\tilde{\Omega}_t$ represent important decisions in the design of the look-ahead model, which we parameterize by θ.

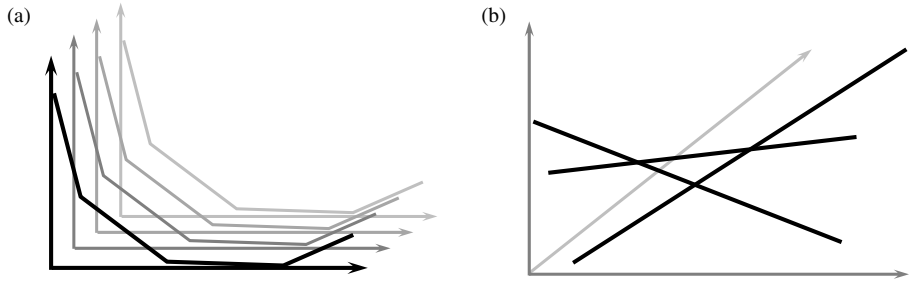
We note that a stochastic look-ahead model is the strategy favored by the stochastic programming community. Equation (31) can be described as a *direct* solution of the look-ahead model, which is to say that we explicitly optimize over all decisions, for each scenario $\tilde{\omega} \in \tilde{\Omega}_t$, all at the same time. It is also possible to solve the look-ahead model using value functions, producing a policy that looks like

$$X_t^\pi(S_t \mid \theta) = \arg\min_{x_{tt}\in\mathcal{X}_t} (C(S_{tt}, x_{tt}) + \mathbb{E}\{\tilde{V}_{t,t+1}(\tilde{S}_{t,t+1}) \mid S_t\}), \tag{32}$$

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) look-ahead model. We emphasize that look-ahead 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 and Pinto [39] and Shapiro et al. [58]).

We need to acknowledge that there are some problems where optimal policies can be found: (s, S) policies are optimal for a special class of inventory problem; myopic policies are optimal for portfolio problems without transaction costs; exact value functions can be found for problems with small, discrete action spaces; and there are some problems where a look-ahead

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



policy is optimal. However, we use the term "function approximation" for three of the four classes of policies because we feel that the vast majority of real applications will not produce an optimal policy.

In addition to these four fundamental classes of policies, we can also create hybrids by mixing and matching. For example, we might use a look-ahead policy with a value function at the end of the look-ahead 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. [17]). Cost function approximations, which include any modification of costs or constraints to achieve a more robust policy, are often combined with a look-ahead policy so that uncertainty in forecasts can be accommodated. For example, we might solve a deterministic approximation of a multiperiod inventory problem but impose lower bounds on inventories to handle uncertainty. These lower bounds represent the tunable parameters of the CFA.

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. [22] 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., Bouzaiene-Ayari et al. [13] and Simão et al. [59]).

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 Powell [42, Chap. 13]), and multidimensional Benders' cuts (b) (see Birge and Louveaux [11] and Shapiro et al. [57]). 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 (16), 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 (see Shapiro [56] for a nice introduction to Monte Carlo sampling methods in the context of stochastic optimization). 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

$$\bar{F}^\pi = \frac{1}{N} \sum_{n=0}^{N} F^\pi(\omega^n). \tag{33}$$

If we are simulating a policy $X^\pi(S_t)$ based on value function approximations, we might write our VFA as

$$\bar{V}_t^\pi(S_t \mid \theta) = \sum_{f \in \mathcal{F}} \theta_f \phi_f^\pi(S_t).$$

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

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

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

Considerable attention has been devoted to the problem of choosing samples carefully. Römisch and Heitsch [24] describes methods for generating scenario trees focusing on the construction of look-ahead models. Bayraksan and Morton [4, 5] provide a nice discussion of assessing solution quality using Monte Carlo methods for stochastic optimization.

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 (16). 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 look ahead), 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, which we refer to as θ, might be the regression parameters of a linear model, the planning horizon in a look-ahead policy, and the number of samples in a scenario tree.

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

$$\min_{\theta \in \Theta^p} \bar{F}^{\pi^p}(\theta) = \frac{1}{N} \sum_{t=0}^{T} C(S_t(\omega^n), X_t^{\pi^p}(S_t(\omega^n) \mid \theta)). \tag{34}$$

This problem has to be solved for each of a (hopefully small) class of policies $\mathcal{P}$ chosen by the designer.

The literature for solving stochastic search problems is quite deep, spanning stochastic search (e.g., Chang et al. [15], Spall [60]), stochastic programming (Birge and Louveaux [11], Shapiro et al. [57]), ranking and selection (Barr and Rizvi [3], Boesel et al. [12]), sample average approximation (Kleywegt et al. [31]), simulation optimization (Andradóttir and Prudius [1], Chick and Gans [16], Chick et al. [23], Fu et al. [20], Hong and Nelson [27], Swisher et al. [62]), and optimal learning (see the review of many techniques in Powell and Ryzhov [43]). 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 (off-line), 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. Look-ahead Policies

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

But not everyone is using a look-ahead model, which means it is not always clear whether the authors are solving the base model or a look-ahead 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 $t=0,\dots,T$, which we typically need to repeat for different sample outcomes. In a look-ahead model, we also need to make decisions over our planning horizon $t'=t,\dots,t+H$, 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 Base Model

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

$$X_t^*(S_t)=\arg\min_{x_t}\left(C(S_t,x_t)+\min_{\pi\in\Pi}\mathbb{E}\left\{\sum_{t'=t+1}^{T}C(S_{t'},X_{t'}^{\pi}(S_{t'}))\,\middle|\,S_t\right\}\right),\tag{35}$$

where $S_{t+1}=S^M(S_t,x_t,W_{t+1})$. What we are doing is finding the best action now given the impact of that action on the rest of the horizon *using the base model*. Note that we did not say that we could compute this policy, but it is a starting point.

The imbedded optimization over policies can look mysterious, so many use an equivalent formulation, which is written

$$X_t^*(S_t)=\arg\min_{x_t}\left(C(S_t,x_t)+\min_{(x_{t'}(\omega),\,t<t'\le T),\,\forall\,\omega\in\Omega_t}\mathbb{E}\left\{\sum_{t'=t+1}^{T}C(S_{t'},x_{t'}(\omega))\,\middle|\,S_t\right\}\right).\tag{36}$$

Instead of writing a policy $X^\pi_{t'}(S_{t'})$, we are writing $x_{t'}(\omega)$, which means that the decision at time t' depends on the sample path that we are following. The problem is that when we specify ω, it means we are identifying the *entire* sample path, from $t=0$ up to $t=T$, which is like being able to know the future.

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 $X^\pi_{t'}(S_{t'})$). 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 (15)). The problem is that since $h_{t'}$ is typically continuous (and multidimensional), $x_{t'}(h_{t'})$ is effectively a continuous function that is the decision we would make if our history is $h_{t'}$ (in other words, a policy); it is pretty to write, but hard to compute.

Both (35) and (36) are look-ahead policies that use the base 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. This is equivalent to computing decisions for *every state* and for *every time period* in the future. The problem, of course, is that we usually cannot compute any of these policies, leading us to consider approximate look-ahead models.

5.2. Building an Approximate Look-ahead Model

To overcome the complexity of solving the exact model, we create what is called a *look-ahead model*, which is an approximation of the base model, which is easier to solve. To distinguish our look-ahead model from the base 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 look-ahead model at time t, with a decision that will be implemented at time t' within the look-ahead model.

There are several strategies that are typically used to simplify look-ahead 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 look-ahead policy that extends one year might be enough to produce high-quality decisions. We can then simulate our policy to produce forecasts of flows over all 10 years.

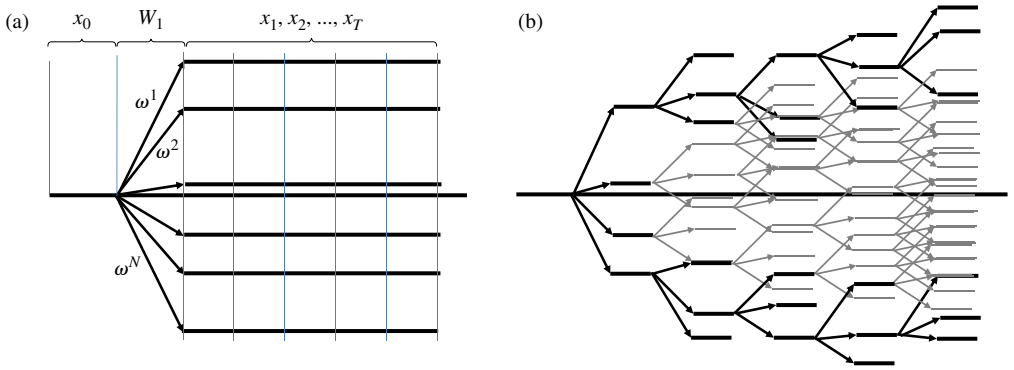
Stage aggregation. A stage represents the process of revealing information followed by the need to make a decision. A common approximation is a two-stage formulation (see Figure 4, panel (a)), where we make a decision x_t, then observe all future events (until $t+H$), and then make all remaining decisions. A more accurate formulation is a multistage model, depicted in Figure 4, panel (b), but these can be computationally very expensive.

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 look ahead, which uses a single point estimate.

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 [46]). 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 $t+1$.

Dimensionality reduction. We may ignore some variables in our look-ahead 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. Whereas we have to track these in the base model (including the evolution of these forecasts), we can hold them fixed in the look-ahead model and then ignore them in the state variable (these become *latent variables*).

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



5.3. A Deterministic Look-ahead Model

A simple deterministic look-ahead model simply uses point forecasts of all exogenous variables, giving us

$$X_t^{LA-D,n}(S_t) = \arg\min_{x_t}\left(C(S_t,x_t) + \sum_{t'=t+1}^{t+H} C(\tilde{S}_{tt'},\tilde{x}_{tt'})\right), \tag{37}$$

where $\tilde{S}_{t,t'+1} = S^M(\tilde{S}_{tt'},\tilde{x}_{tt'},\bar{W}_{tt'})$, and where $\bar{W}_{tt'} = \mathbb{E}\{W_{tt'} \mid S_t\}$ is a forecast of $W_{t'}$ made at time t. These models are often referred to as rolling horizon procedures or model predictive control, but these terms can also be applied to stochastic approximations of the look-ahead model. However, deterministic approximations are most widely used in practice.

5.4. A Stochastic Look-ahead Model

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

$$X_t^{LA-SP,n}(S_t^n) = \arg\min_{x_t}\left(C(S_t^n,x_t) + \min_{(\tilde{x}_{tt'}(\tilde{\omega}),\, t<t'\le t+H),\,\forall\tilde{\omega}\in\tilde{\Omega}_t^n} \tilde{\mathbb{E}}^n\left\{\sum_{t'=t+1}^{t+H} C(\tilde{S}_{tt'},\tilde{x}_{tt'}(\tilde{\omega}))\,\middle|\, S_t\right\}\right). \tag{38}$$

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

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

In (38), the expectation $\tilde{\mathbb{E}}^n\{\cdot \mid S_t\}$ 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 at time t'), 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 look-ahead 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, panel (a) or (b)), and 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 (38) is so large that we have to turn to decomposition methods that solve the problem within a tolerance (see Birge and Louveaux [11] and Rockafellar and Wets [48]). However, our notation now makes it clear that we are solving a look-ahead model. Bounds on the performance of a look-ahead model do not translate to bounds on the performance of the policy in terms of its ability to minimize the base objective function (16).

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

$$X_t^{VFA,n}(S_t) = \arg\min_{x_t}\big(C(S_t,x_t) + \mathbb{E}\big\{\bar{V}_{t+1}^{n-1}(S_{t+1}) \mid S_t, x_t\big\}\big), \tag{39}$$

where

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

Here, we are using some sort of functional approximation $\bar{V}_{t+1}^{n-1}(S_{t+1})$, 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 base model, not the look-ahead model. If we have access to a convenient post-decision state S_t^x, we can drop the expectation and use

$$X_t^{VFA,n}(S_t) = \arg\min_{x_t}\big(C(S_t,x_t) + \bar{V}_t^{n-1}(S_t^x)\big), \tag{40}$$

where the post-decision value function approximation (40) is different than the predecision value function approximation in (39).

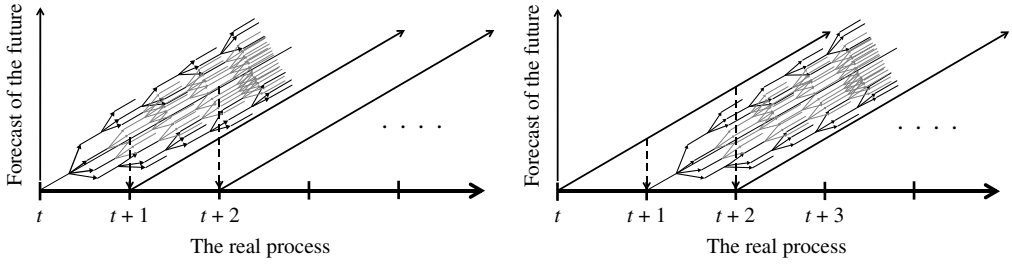
We note that once we have the value function approximations $\bar{V}_t(S_t)$ (or $\bar{V}_t(S_t^x)$) for $t = 0,\ldots,T$, we have a complete policy for the base model, where we assume that the approximation $\bar{V}_t(S_t)$ gives us a value for every possible state (which means we have a decision for every possible state). By contrast, the look-ahead 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 look-ahead policy, we produce a series of decisions $\tilde{x}_{tt'}(\tilde{S}_{tt'})$ that can be interpreted as a policy within the look-ahead model. But because it is only for a small sample of states $\tilde{S}_{tt'}$ (that is, the states that appear in the scenario tree), 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 (38) from scratch.

The need to recompute the look-ahead model is not limited to deterministic approximations or approximations of stochastic models through scenario trees. We might solve the look-ahead model using a value function, as we did previously in Equation (32), repeated here for convenience:

$$X_t^{\pi}(S_t) = \arg\min_{x_{tt}\in\mathcal{X}_t}\big(C(S_{tt},x_{tt}) + \mathbb{E}\big\{\tilde{V}_{t,t+1}(\tilde{S}_{t,t+1}) \mid S_t\big\}\big). \tag{41}$$

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 look-ahead model (that is why we have a tilde over the V). Because the look-ahead 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 $\bar{V}_t(S_t)$ that is an approximation of the downstream value in the base model.

FIGURE 5. Illustration of rolling horizon procedure, using a stochastic model of the future (from Powell et al. [44]).



5.5. Evaluating a Look-ahead Policy

The correct way to evaluate a look-ahead policy (or any policy) is in the context of the objective function for the base model given in (16). Figure 5 illustrates the process of solving a stochastic look-ahead model using scenario trees, then stepping forward in time to simulate the performance of our look-ahead policy. As of this writing, there is a *lot* of work evaluating the quality of the solution of a look-ahead model (which can be quite hard to solve) but very little evaluating the performance of a look-ahead policy in terms of its ability to solve the base model (see, for example, Bayraksan and Morton [5], Mak et al. [34], Philpott and Guan [41] and Rockafellar and Wets [48]).

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 and Bordons [14] for a good introduction to model predictive control and Lewis et al. [33] for a modern introduction to approximate dynamic programming and optimal control).

It has been our experience that although many researchers in stochastic programming understand that a stochastic program should be evaluated in a "simulator," there is a fairly widespread lack of appreciation that the simulator is actually a way of approximating the objective function (16), which is the real problem we are trying to solve (see Defourny et al. [17], McCusker et al. [37], Mulvey et al. [38], Takriti et al. [63], van der Weijde and Hobbs [65] for a few examples). For example, Ben-Tal et al. [8] propose a policy using a robust (min-max) look-ahead policy, which they then evaluate by averaging a series of simulations, which means using the objective in (16) (see Ben-Tal et al. [8, §4.3]). The lack of consistency between the objective function in the simulator (the base model) and that used in the look-ahead model reflect, in our view, a fairly widespread misunderstanding that the simulator is actually the stochastic analog of the objective function (4) used in deterministic models.

5.6. Comments

The distinction between base models and look-ahead models has not entered the literature, so care has to be used when deciding if a researcher is solving the base model or just a look-ahead model. It is our experience that the vast majority of papers using stochastic programming for multistage problems are using look-ahead policies (see, for example, Dupačová et al. [19], Jacobs et al. [28], Jin et al. [29], Takriti et al. [63], Wallace and Fleten [67]). 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. [58] formulate and solve 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. This strategy has come to be known as the *stochastic dual decomposition procedure* (or SDDP, first introduced by Pereira and Pinto [39]). Although this could be viewed as a 120-period

look-ahead model, it is perfectly reasonable to define the 120-period stochastic program as the base model. This is possible because after he solves his 120-period problem, he has a series of Benders' cuts that defines a policy *for every state, for every time period*. This is possible because he makes an assumption known in this community as *intertemporal independence*, which means that after making a decision, the information process refreshes itself and does not retain history (intertemporal independence simply means that we can ignore the information process in the post-decision state). As a result, it is only necessary to compute one set of cuts for every time period, rather than one for every node in a scenario tree. However, SDDP requires approximating the underlying information process using a finite (and small) set of sample realizations; the bounds on the quality of the solution are only bounds for this approximate problem (see Philpott and Guan [41, §2] and Shapiro et al. [58, §3.1]).

Our own work (Topaloglu and Powell [64], Powell et al. [45], Simão et al. [59] for example) also uses value functions that are approximations of the base model, over the entire horizon. In both sets of research, once the value functions (Benders' cuts in Shapiro et al. [58] and Sen and Zhou [53]) 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 base model rather than a look-ahead model.

Scenario trees always imply a look-ahead policy, because stepping forward in the base model will almost invariably put us in a state that is not represented in the scenario tree. Sen and Zhou [53] introduce multistage stochastic decomposition (MSD) building on the breakthrough of stochastic decomposition for two-stage problems (Higle and Sen [25]). The MSD algorithm generates scenarios that asymptotically cover the entire state space, but any practical implementation (which is limited to a finite number of samples) would still require reoptimizing 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 look-ahead model may still be quite difficult, and a substantial literature has grown around the problem of solving stochastic look-ahead models (Birge and Louveaux [11], Higle and Sen [26], King and Wallace [30], Rockafellar and Wets [48], Römisch and Heitsch [24]). 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 the following:

- An optimal solution to a look-ahead model is, with rare exceptions, not an optimal policy.
- A bound on a look-ahead model is not a bound on the performance of the resulting policy (with rare exceptions, such as Shapiro et al. [58]).

For an in-depth treatment of the properties of look-ahead policies, see the work of Sethi (see Bhaskaran and Sethi [10], Sethi and Bhaskaran [54], and Sethi and Haurie [55]). 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 look-ahead policy for solving a dynamic program, and that what we care most about is the performance of the policy for solving the base model (that is, Equation (34)) rather than how well we solve the look-ahead model (Equation (31)). This comment also ignores the fact that the look-ahead 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 look-ahead model (see Jacobs et al. [28] and Shapiro et al. [57] for examples). We feel that placing stochastic programming within the broader framework of dynamic programming separates the look-ahead model from the base 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 look-ahead objective function.

6. Direct Policy Search vs. Bellman Error Minimization

Look-ahead 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

$$X_t^\pi(S_t \mid \theta) = \arg\min_{x \in \mathcal{X}_t} \left(C(S_t, x) + \gamma \sum_{f \in \mathcal{F}} \theta_f \phi_f(S_t^x) \right). \tag{42}$$

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 (34) using any of a range of stochastic search methods (see Maxwell et al. [36, §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

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

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 (42) and then simulating this policy from time t until the end of the horizon following sample path ω^n using

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

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 back propagation in the early work of Paul Werbos (Werbos [68, 69, 70]), who introduced the idea of backward differentiation. It is beyond the scope of our presentation to discuss the merits of these two approaches (see Powell [42, Chapters 9 and 10]), 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. Although 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 and Parr [32].

(3) Least squares approximate policy iteration using instrumental variables (or projected Bellman error minimization), also described in Lagoudakis and Parr [32] and implemented in Scott et al. [52].

(4) Direct policy search, using the same basis functions used in the testing with least squares policy iteration (Scott et al. [52], using Scott et al. [51] to do the stochastic search).

(5) Approximate value iteration with a backward pass (known as TD(1) in the reinforcement learning community (Sutton and Barto [61]) or back propagation in the controls community (Werbos [70])), but using a piecewise linear value function (in the energy resource variable), which maintains concavity (see Salas and Powell [50]).

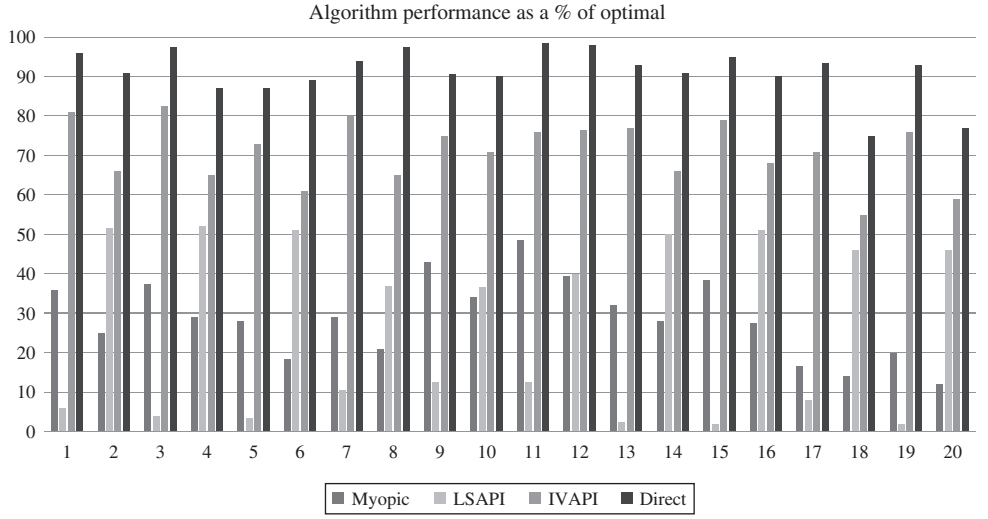
(6) Approximate policy iteration (API)—Pham [40] evaluated API using linear models, support vector regression, Gaussian process regression, tree regression, kernel regression, and a local parametric method on a library of test problems with optimal solutions.

The first four problems were run on a steady state (infinite horizon) problem, whereas 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% of optimality (with a discount factor of 0.9999, 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 data sets were also evaluated by Pham [40], 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 percentage of the optimal. The runs using direct policy search were almost all over 90% of optimality, whereas 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% of optimality on the finite horizon problems (Salas and Powell [50]).

These observations need to be accompanied by a few caveats. The use of the piecewise linear value function approximation (used in Salas and Powell [50]) 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 that 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).

FIGURE 6. The performance of a series of approximation algorithms relative to the optimal solution for 20 benchmark storage problems.



Note. These include a myopic policy, a basic form of least squares approximate policy iteration, least squares approximate policy iteration using instrumental variables (IVAPI), and direct policy search using the same structure for the policy (from Scott et al. [52]).

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 and Powell [50]) but surprisingly poorly (e.g., around 70% of optimal) using simple, quadratic 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% of optimal). However, this is limited to low-dimensional parameter vectors, which exclude problems where the regression vector θ is a function of time.
- Approximate value iteration when exploiting convexity works best of all (over 98% of optimal), but it 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 and Barto [61]).
- A deterministic look-ahead policy underperformed approximate value iteration but performed as well as policy search (approximately 95%–99% of optimal). The look-ahead 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 worked adequately. Our best results on the storage problems, using support vector regression, might get over 80% of optimal (sometimes to 90%) 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 (29) using direct policy search (see Maxwell et al. [35] for a good example of this). However, if you are choosing θ to tune the policy in (29), 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, since we are not fitting the approximation to the value of being in a state.

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

$$\begin{aligned} X^{\mathrm{CFA}}(S_t \mid \theta) &= \arg\min_{x_t} \bar{C}^{\pi}(S_t, x_t \mid \theta) \\ &= \arg\min_{x_t}\left(C(S_t, x_t) + \sum_{f\in\mathcal{F}}\theta_f\phi_f(S_t^x)\right). \end{aligned} \tag{43}$$

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 (43). In our energy storage example, we used basis functions such as $S_t x_t$, 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.

Our belief is that cost function approximations are widely used in engineering practice, but without being identified as such. The storage example described in this setting represents an instance of a problem where a CFA (constructed using a simple, additive correction term) works extremely well. We suspect that there is no shortage of common sense approaches that can be used to make a policy based on solving some deterministic model work better. We offer that the construction of these approximations is exactly analogous to the steps involved in constructing value function approximations or policy function approximations. All that has been missing is the idea that this strategy should be recognized as a completely valid approach.

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 look-ahead policies, policies based on value function approximations, or policy function approximations. Deterministic look-ahead policies that have been tweaked to produce more robust solutions (such as introducing safety stocks or reserve capacities) should be viewed as look-ahead cost function approximations. Cost function approximations are widely used in practice but are generally dismissed as some sort of heuristic compared with more elegant and "sophisticated" policies based on stochastic look aheads (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 a priori reason why a cost function approximation, policy function approximation, policy based on value function approximation, or a look ahead that 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 mathematical programming algorithm designed to handle constraints.
- 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 look-ahead model but are hard to approximate using a statistical model.

• Look-ahead 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 look-ahead 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 look-ahead model should always be compared to the performance of a deterministic look-ahead model. Just because the underlying problem is stochastic does not mean that a deterministic look-ahead model will not work.

Hybrid policies can be particularly valuable. Instead of building a look-ahead 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 (look-ahead 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).

Acknowledgments

The author warmly acknowledges the helpful comments of the two reviewers and managing editor. This research was supported by the National Science Foundation [Grant CMMI-0856153].

References

[1] S. Andradóttir and A. A. Prudius. Adaptive random search for continuous simulation optimization. *Naval Research Logistics* 57(6):583–604, 2010.

[2] C. Bandi and D. J. Bertsimas. Tractable stochastic analysis in high dimensions via robust optimization. *Mathematical Programming Series B* 134(1):23–70, 2012.

[3] D. R. Barr and M. H. Rizvi. An introduction to ranking and selection procedures. *Journal of American Statistical Association* 61(315):640–646, 1966.

[4] G. Bayraksan and D. P. Morton. Assessing solution quality in stochastic programs. *Mathematical Programming Series B* 108(2–3):495–514, 2006.

[5] G. Bayraksan and D. P. Morton. Assessing solution quality in stochastic programs via sampling. M. R. Oskoorouchi, ed. *Tutorials in Operations Research*. INFORMS, Hanover, MD, 102–122, 2009.

[6] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[7] A. Ben-Tal, L. E. Ghaoui, and A. Nemirovski. *Robust Optimization*. Princeton University Press, Princeton, NJ, 2009.

[8] A. Ben-Tal, B. Golany, A. Nemirovski, and J.-P. Vial. Retailer-supplier flexible commitments contracts: A robust optimization approach. *Manufacturing & Service Operations Management* 7(3):248–271, 2005.

[9] H. Beyer and B. Sendhoff. Robust optimization—A comprehensive survey. *Computer Methods in Applied Mechanics and Engineering* 196(33–34):3190–3218, 2007.

[10] S. Bhaskaran and S. P. Sethi. Decision and forecast horizons in a stochastic environment: A survey. *Optimal Control Applications and Methods* 8(3):201–217, 1987.

[11] J. R. Birge and F. Louveaux. *Introduction to Stochastic Programming*, 2nd ed., Springer, New York, 2011.

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

[13] B. Bouzaiene-Ayari, C. Cheng, S. Das, R. Fiorillo, and W. B. Powell. From single commodity to multiattribute models for locomotive optimization: A comparison of integer programming and approximate dynamic programming. *Transportation Sci.*, ePub ahead of print July 28, 2014, http://dx.doi.org/10.1287/trsc.2014.0536.

[14] E. Camacho and C. Bordons. *Model Predictive Control*. Springer, London, 2004.

[15] H. S. Chang, M. C. Fu, J. Hu, and S. I. Marcus. *Simulation-Based Algorithms for Markov Decision Processes*. Springer, Berlin, 2007.

[16] S. E. Chick and N. Gans. Economic analysis of simulation selection problems. *Management Science* 55(3):421–437, 2009.

[17] B. Defourny, D. Ernst, and L. Wehenkel. Scenario trees and policy selection for multistage stochastic programming using machine learning. *INFORMS Journal on Computing* 25(3):488–501, 2013.

[18] R. Dorfman. The discovery of linear programming. *Annals of the History of Computing* 6(3):283–295, 1984.

[19] J. Dupačová, G. Consigli, and S. Wallace. Scenarios for multistage stochastic programs. *Annals of Operations Research* 100(1–4):25–53, 2000.

[20] M. C. Fu, F. Glover, and J. April. Simulation optimization: A review, new developments, and applications. M. E. Kuhl, N. M. Steiger, F. B. Armstrong, J. A. Joines, eds., *Proceedings of the 37th Conference on Winter Simulation*, IEEE, Piscataway, NJ, 83–95, 2005.

[21] J. Gittins, K. Glazebrook, and R. R. Weber. *Multi-Armed Bandit Allocation Indices*. John Wiley & Sons, Chichester, UK, 2011.

[22] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer, New York, 2009.

[23] D. He, S. E. Chick, and C.-H. Chen. 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* 37(5):951–961, 2007.

[24] H. Heitsch and W. Römisch. Scenario tree modeling for multistage stochastic programs. *Mathematical Programming* 118(2):371–406, 2009.

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

[26] J. Higle and S. Sen. *Stochastic Decomposition: A Statistical Method for Large Scale Stochastic Linear Programming*. Kluwer Academic Publishers, Boston, 1996.

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

[28] J. Jacobs, G. Freeman, J. Grygier, D. P. Morton, G. Schultz, K. Staschus, and J. Stedinger. SOCRATES: A system for scheduling hydroelectric generation under uncertainty. *Annals of Operational Research* 59(1):99–133, 1995.

[29] S. Jin, S. Ryan, J. Watson, and D. Woodruff. Modeling and solving a large-scale generation expansion planning problem under uncertainty. *Energy Systems* 2(3–4):209–242, 2011.

[30] A. J. King and S. Wallace. *Modeling with Stochastic Programming*. Springer-Verlag, New York, 2012.

[31] A. J. Kleywegt, A. Shapiro, and T. Homem-de-Mello. The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization* 12(2):479–502, 2002.

[32] M. Lagoudakis and R. Parr. Least-squares policy iteration. *Journal of Machine Learning Research* 4:1107–1149, 2003.

[33] F. L. Lewis, D. Vrabie, and V. L. Syrmos. *Optimal Control*, 3rd ed. John Wiley & Sons, Hoboken, NJ, 2012.

[34] W.-K. Mak, D. P. Morton, and R. Wood. Monte Carlo bounding techniques for determining solution quality in stochastic programs. *Operations Research Letters* 24(1–2):47–56, 1999.

[35] M. S. Maxwell, S. G. Henderson, and H. Topaloglu. Tuning approximate dynamic programming policies for ambulance redeployment via direct search. *Stochastic Systems* 3(2):322–361, 2013.

[36] M. S. Maxwell, M. Restrepo, S. G. Henderson, and H. Topaloglu. Approximate dynamic programming for ambulance redeployment. *INFORMS Journal on Computing* 22(2):266–281, 2010.

[37] S. A. McCusker, B. F. Hobbs, and Y. Ji. Distributed utility planning using probabilistic production costing and generalized Benders decomposition. *IEEE Transactions on Power Systems* 17(2):497–505, 2002.

[38] J. M. Mulvey, R. J. Vanderbei, and S. A. Zenios. Robust optimization of large-scale systems. *Operations Research* 43(2):264–281, 1995.

[39] M. V. F. Pereira and L. M. V. G. Pinto. Multi-stage stochastic optimization applied to energy planning. *Mathematical Programming* 52(2):359–375, 1991.

[40] T. Pham. Experiments with approximate policy iteration. Ph.D. thesis, Princeton University, Princeton, NJ, 2013.

[41] A. Philpott and Z. Guan. On the convergence of stochastic dual dynamic programming and related methods. *Operations Research Letters* 36(4):450–455, 2008.

[42] W. B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*, 2nd ed. John Wiley & Sons, Hoboken, NJ, 2011.

[43] W. B. Powell and I. O. Ryzhov. *Optimal Learning*. John Wiley & Sons, Hoboken, NJ, 2012.

[44] W. B. Powell, H. P. Simão, and B. Bouzaiene-Ayari. Approximate dynamic programming in transportation and logistics: A unified framework. *EURO Journal on Transportation and Logistics* 1(3):237–284, 2012.

[45] W. B. Powell, A. George, H. P. Simão, W. R. Scott, A. D. Lamont, and J. Stewart. SMART: A stochastic multiscale model for the analysis of energy resources, technology, and policy. *INFORMS Journal on Computing* 24(4):665–682, 2012.

[46] M. L. Puterman. *Markov Decision Processes*, 2nd ed. John Wiley & Sons, Hoboken, NJ, 2005.

[47] R. T. Rockafellar and S. Uryasev. The fundamental risk quadrangle in risk management, optimization, and statistical estimation. *Surveys in Operations Research and Management Science* 18(1):33–53, 2013.

[48] R. T. Rockafellar and R. J.-B. Wets. Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of Operations Research* 16(1):119–147, 1991.

[49] A. Ruszczyński. Risk-averse dynamic programming for Markov decision processes. *Mathematical Programming* 125(2):235–261, 2010.

[50] D. F. Salas and W. B. Powell. Benchmarking a scalable approximate dynamic programming algorithm for stochastic control of multidimensional energy storage problems. Technical Report 2004, Department of Operations Research and Financial Engineering, Princeton University, Princeton, NJ, 2013.

[51] W. R. Scott, P. Frazier, and W. B. Powell. The correlated knowledge gradient for simulation optimization of continuous parameters using Gaussian process regression. *SIAM Journal on Optimization* 21(3):996–1026, 2011.

[52] W. R. Scott, W. B. Powell, and S. Moazeni. Least squares policy iteration with instrumental variables vs. direct policy search: Comparison against optimal benchmarks using energy storage. Technical report, Dept. of Operations Research and Financial Engineering, Princeton University, Princeton, NJ, 2013.

[53] S. Sen and Z. Zhou. Multistage stochastic decomposition: A bridge between stochastic programming and approximate dynamic programming. *SIAM Journal on Optimization* 24(1): 127–153, 2014.

[54] S. P. Sethi and S. Bhaskaran. Conditions for the existence of decision horizons for discounted problems in a stochastic environment. *Operations Research Letters* 4(2):61–64, 1985.

[55] S. P. Sethi and A. Haurie. Decision and forecast horizons, agreeable plans, and the maximum principle for infinite horizon control problem. *Operations Research Letters* 3(5):261–265, 1984.

[56] A. Shapiro. Monte Carlo sampling methods. A. Ruszczyński and A. Shapiro, eds. *Stochastic Programming*, Handbook in Operations Research & Management Science, Vol. 10. North-Holland, Amsterdam, 353–425, 2003.

[57] A. Shapiro, D. Dentcheva, and A. Ruszczyński. *Lectures on Stochastic Programming: Modeling and Theory*. SIAM, Philadelphia, 2009.

[58] A. Shapiro, W. Tekaya, J. Paulo da Costa, and M. F. Pereira. Risk neutral and risk averse stochastic dual dynamic programming method. *European Journal of Operational Research* 224(2):375–391, 2013.

[59] H. P. Simão, J. Day, A. R. George, T. Gifford, J. Nienow, and W. B. Powell. An approximate dynamic programming algorithm for large-scale fleet management: A case application. *Transportation Science* 43(2):178–197, 2009.

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

[61] R. S. Sutton and A. G. Barto. *Reinforcement Learning*, Vol. 35. MIT Press, Cambridge, MA, 1998.

[62] J. R. Swisher, P. D. Hyden, S. H. Jacobson, and L. W. Schruben. A survey of simulation optimization techniques and procedures. *Proceedings of the Winter Simulation Conference*, 119–128, IEEE, Piscataway, NJ, 2000.

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

[64] H. Topaloglu and W. B. Powell. Dynamic-programming approximations for stochastic time-staged integer multicommodity-flow problems. *INFORMS Journal on Computing* 18(1):31–42, 2006.

[65] A. H. van der Weijde and B. F. Hobbs. The economics of planning electricity transmission to accommodate renewables: Using two-stage optimisation to evaluate flexibility and the cost of disregarding uncertainty. *Energy Economics* 34(6):2089–2101, 2012.

[66] R. J. Vanderbei. *Linear Programming: Foundations and Extensions*. Kluwer Academic Publishers, Boston, 1996.

[67] S. W. Wallace and S.-E. Fleten. Stochastic programming models in energy. A. Ruszczyński and A. Shapiro, eds. *Stochastic Programming*, Vol. 10. Elsevier Science B.V., Amsterdam, 637–677, 2003.

[68] P. J. Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. Ph.D. thesis, Harvard University, Cambridge, MA, 1974.

[69] P. J. Werbos. Backpropagation and neurocontrol: A review and prospectus. International Joint Conference on Neutal Networks, 209–216, IEEE, Piscataway, NJ, 1989.

[70] P. J. Werbos. Approximate dynamic programming for real-time control and neural modelling. D. J. White and D. A. Sofge, eds. *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*, 493–525, 1992.