EURO J Transp Logist (2012) 1:237–284
DOI 10.1007/s13676-012-0015-8

TUTORIAL

Approximate dynamic programming in transportation and logistics: a unified framework

Warren B. Powell · Hugo P. Simao · Belgacem Bouzaiene-Ayari

Received: 10 March 2012 / Accepted: 3 August 2012 / Published online: 30 August 2012

Abstract Deterministic optimization has enjoyed a rich place in transportation and logistics, where it represents a mature field with established modeling and algorithmic strategies. By contrast, sequential stochastic optimization models (dynamic programs) have been plagued by the lack of a common modeling framework, and by algorithmic strategies that just do not seem to scale to real-world problems in transportation. This paper is designed as a tutorial of the modeling and algorithmic framework of approximate dynamic programming; however, our perspective on approximate dynamic programming is relatively new, and the approach is new to the transportation research community. We present a simple yet precise modeling framework that makes it possible to integrate most algorithmic strategies into four fundamental classes of policies, the design of which represents approximate solutions to these dynamic programs. The paper then uses problems in transportation and logistics to indicate settings in which each of the four classes of policies represents a natural solution strategy, highlighting the fact that the design of effective policies for these complex problems will remain an exciting area of research for many years. Along the way, we provide a link between dynamic programming, stochastic programming and stochastic search.

Keywords Approximate dynamic programming · Dynamic programming · Stochastic programming · Stochastic search · Logistics · Dynamic vehicle routing

Introduction

The transportation science community has long recognized the power of mathematical programming. Indeed, problems in transportation and logistics served as the

W. B. Powell (✉) · H. P. Simao · B. Bouzaiene-Ayari
Department of Operations Research and Financial Engineering,
Princeton University, Princeton, NJ 08544, USA
e-mail: powell@princeton.edu

original motivating application for much of the early work in math programming (Dantzig 1951; Ferguson and Dantzig 1955). At the same time, while George Dantzig has received considerable (and well-deserved) recognition for introducing the simplex algorithm, it can be argued that his greatest contribution was establishing the fundamental *modeling* framework which is widely written as:

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

subject to

$$Ax = b, \tag{2}$$

$$x \leq u, \tag{3}$$

$$x \geq 0. \tag{4}$$

Of course, the need for integrality constraints created a long and highly successful search for general purpose tools for solving integer programs, but there are many problems that still require the use of metaheuristics to obtain effective algorithms (Taillard et al. 1997; Le Bouthillier and Crainic 2005; Braysy and Gendreau 2005). Regardless of the algorithmic strategy, however, this community uniformly uses Dantzig's fundamental modeling framework given by Eqs. (1)–(4), which has become a unifying language that allows people to share and compare algorithmic strategies. As of this writing, the transportation science community has developed considerable maturity in translating complex operational problems in transportation and logistics into this modeling framework.

The situation changes dramatically when we introduce uncertainty. From the beginning, Dantzig recognized the need to capture uncertainty in his work on aircraft scheduling (Dantzig and Ferguson 1956), which helped to motivate the emerging field of stochastic programming. But stochastic programming has become just one of a family of algorithmic strategies in a sea of research that is characterized by a relative lack of standard notational systems. At around the same time, Bellman (1957) developed his seminal work on dynamic programming, producing modeling and algorithmic strategies that appeared to have no overlap whatsoever with Dantzig's early work on stochastic programming.

The lack of a common modeling framework has complicated the process of understanding and comparing algorithmic strategies. As of this writing, researchers do not even agree on what objective function is being evaluated. In a deterministic problem, if solutions x^1 and x^2 both satisfy the constraints (2)–(4), then x^1 is better than x^2 if $cx^1 < cx^2$. The situation in stochastic optimization is much more subtle.

Our interest is primarily in *sequential* optimization problems, otherwise known as dynamic programs. Deterministic versions of these problems might be written

$$\min_{x_0,\ldots,x_T} \sum_{t=0}^{T} c_t x_t \tag{5}$$

subject to

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

$$x_t \leq u_t, \tag{7}$$

$$x_t \geq 0. \tag{8}$$

Problems of this form have been called "dynamic" in the past, but we prefer the term "time-dependent." Now consider what happens when we introduce uncertainty. For example, we may have to deal with random demands, random travel times, and equipment failures. Bellman introduced the term *dynamic program* to refer to problems in which information evolves over time, to distinguish it from Dantzig's term "math program" which was gaining popularity at the time. Bellman introduced his now famous optimality equation

$$V_t(S_t) = \min_{a \in \mathcal{A}} \left(C(S_t, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|S_t, a) V_{t+1}(s') \right). \tag{9}$$

Equation (9) became synonymous with the new field of dynamic programming. Unfortunately, Eq. (9) is not a dynamic program in the same way that Eqs. (1)–(4) represent a math program. Equation (9) is a method for *solving* a dynamic program, just as the simplex algorithm is a way of solving a linear program. This subtle but important distinction has contributed to the confusion that surrounds stochastic versions of sequential optimization problems.

One goal of this paper is to unify the many modeling and algorithmic strategies that have emerged in stochastic optimization, focusing on the contextual domain of operational problems in transportation and logistics. It is not enough, however, to simply create a modeling framework. Dantzig's contribution of the simplex algorithm was to show that the framework in (1)–(4) produces a *solvable* model. Just as a family of algorithmic strategies have emerged around Eqs. (1)–(4) for deterministic optimization problems, we provide a compact framework for describing a very broad class of *computable* algorithmic strategies for stochastic optimization problems, illustrated using specific applications in transportation. Just as we have learned to identify the strategies that are most likely to work on a specific integer programming problem, we hope to help identify the best algorithmic strategies for stochastic optimization problems arising in transportation.

This paper is designed as a tutorial, but in the process we are going to present a fresh perspective of how different stochastic optimization problems in transportation can be presented with a compact, elegant modeling framework. Our framework is based on classical ideas in control theory, but is not familiar to the transportation science community. The purpose of the paper is not to argue for a particular algorithmic strategy, since we believe that problems in transportation are so diverse that no single strategy will work for all problems. However, we do argue that the universe of algorithms can be reduced to four fundamental classes of policies. In the paper, we present problems that are suited to each of these four classes, and then demonstrate how more complex problems can be solved using hybrids.

We begin our presentation with a description of how to model stochastic, dynamic programs in "Modeling". While the transportation science community enjoys considerable maturity in the modeling of deterministic math programming models, there is a wide diversity of modeling strategies when we make the transition to stochastic, dynamic programs. The section "Policies" then describes four fundamental classes of policies that have been used to solve different problem classes. The section "Linking with other communities" takes a detour to establish links between stochastic search, stochastic programming and dynamic programming. We next use a variety of problems from transportation and logistics to illustrate each of these policies. These include stochastic shortest path problems (see "Shortest path problems"); different classes of inventory problems, including spatially distributed problems that arise in fleet management (see "Inventory problems"); the load matching problem of truckload trucking (see "Fleet management problems"); and a version of the dynamic vehicle routing problem (see "Dynamic vehicle routing"), which we use as an example of a genuinely hard problem which remains an active area of research. The section "Choosing a policy" discusses some general guidelines for choosing a policy, and "Concluding remarks" concludes the paper.

Modeling

We cannot have a productive discussion about algorithms for solving a model until we can agree on how to model a problem. While our community has developed considerable skill and maturity in modeling deterministic optimization problems, we lack a consensus on how to model stochastic optimization problems. Complicating the situation is the tendency to model a problem with an algorithmic strategy in mind. We prefer a modeling style, similar to that used in deterministic math programming, where the representation of the problem is separate from the choice of algorithmic strategy.

There are five fundamental elements of any sequential (multistage) stochastic optimization model: states, actions, exogenous information, the transition function and the objective function (see Powell 2011, Chap. 5, available at http://adp.princeton.edu). We specifically avoid any discussion of how to make a decision, which is addressed in the section "Policies". Our presentation of these elements is somewhat high level. The best way to understand how to use them is by example, and we do this in the remainder of the paper.

It is important to recognize that while we use the vocabulary of dynamic programming, our representation is conceptually the same as a stochastic programming *model*, as long as it is not confused with a stochastic programming *algorithm*. The bigger difference lies in the algorithmic strategies used for solving the dynamic program. Below, we show that that theses differences are simply different classes of policies, and should really be viewed as part of a larger family.

The state variable

The academic community has had a surprisingly difficult time defining a state variable. Bellman's (1957, p. 81) seminal text introduced a state variable with "... we have a physical system characterized at any stage by a small set of parameters, the *state variables*". Puterman's (2005, p. 18) more modern introduction to dynamic programming introduces state variables with "At each decision epoch, the system occupies a *state*". We prefer the definition offered in Powell (2011):

Definition 1.1 The *state variable* S_t is the minimally dimensioned function of history that is necessary and sufficient to compute the decision function, the transition function, and the cost function.

In other words, the state variable is all the information you need (and only the information you need) to model the problem from any given point in time onward. However, even with a definition such as this, it takes time to learn how to identify the information that makes up a state variable. The applications that we address in the rest of the paper serve as an initial set of examples.

There are three categories of state variables:

- The resource state R_t: It can also be convenient to call this the physical state, but "resource state" works well in transportation and logistics. This describes the location of trucks, trains and planes, the amount of inventory on hand, the status of drivers, pilots and equipment operators, and the set of all customer requests or demands (passengers to be moved, loads of freight, requests for product).
- The information state I_t: This includes information other than resources (which technically is a form of information) that is needed to make a decision, compute the transition function or compute the cost/contribution function. The information state might include prices and weather.
- The knowledge state K_t: Also known as the belief state, the knowledge state arises when there is uncertainty about distributions of random variables, where observations of random variables from exogenous sources may change our distribution of belief about a random variable. For example, the time to traversing a link in a network may be random; we may not know the distribution of these times, but we have an estimate. The time that we observe by traversing the link may be used to update our belief about the probability distribution of times.

We primarily focus on problems which involve the management of resources. For this reason, there are many problems where R_t is the only state variable. More often, we encounter problems with information that is not a resource (such as prices) which we include in I_t. Below, we also illustrate applications where the knowledge state K_t comes into play. When this is the case, we write our state variable as $S_t = (R_t, I_t, K_t)$.

In transportation, there are many applications that involve what are called *lagged* processes, which capture the fact that we may know something now (at time t) about the future (at time $t' > t$). Examples include:

$D_{tt'}$ = A demand to be served at time t' that is known at time t,

$f^D_{tt'}$ = A forecast of demand at time t' that we make at time t,
$R_{tt'}$ = A resource (such as a truck, locomotive, aircraft or pilot) that is known at time t to be available at time t'. Often, t' would be known as the estimated time of arrival in the context of equipment that is moving and will arrive at time t',
$x_{tt'}$ = A decision (say, to assign a driver to pick up a delivery) made at time t to be implemented at time t'.

We refer to this as "(t, t')" notation, and it is easy to see that there are numerous applications in transportation. We will often write these variables as vectors, as in $D_t = (D_{tt'})_{t' \geq t}$.

Later in the paper, we are going to take advantage of a concept known as a post-decision state (see Powell 2011 [Chap. 4] for a more thorough discussion of post-decision states). This is the state of the system immediately after we have made a decision, but before any new information has arrived. As a result, this is measured at time t, so we designate it by S^x_t (or S^a_t if we are using action a). For example, imagine an inventory problem where R_t is the amount of inventory at time t, x_t is an order that arrives instantaneously, and $\hat{D}_{t+1}$ is the demand that occurs between t and $t+1$. We might write the evolution of the inventory using:

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

The post-decision resource state for this problem is given by $R^x_t = R_t + x_t$. If we sell our product at a price p_t that evolves according to $p_{t+1} = p_t + \hat{p}_{t+1}$, then our post-decision state $S^x_t = (R^x_t, p_t)$. The concept of the post-decision state appears to be particularly useful in the context of transportation and logistics.

The decision variable

We find it useful to use two notational systems for decision variables. We use x_t primarily to represent decisions in the form of vectors, while we let a_t represent scalar, discrete actions. Discrete, scalar actions arise when we are making decisions about a single driver, a single truck or a single aircraft. Both problem classes are important, in some cases at the same time. Some algorithmic strategies in stochastic optimization are specifically designed for discrete action spaces (without any assumption of convexity), while others are well suited for vector valued actions.

When modeling a problem, it is important to define decision variables and how they impact the problem. However, we do not address the issue of how to make decisions, other than to define a *decision function* which we denote $X^\pi(S_t)$ for making decisions x_t, or $A^\pi(S_t)$ for actions a_t. The decision function $X^\pi(S_t)$ (or $A^\pi(S_t)$) is known in the dynamic programming community as a *policy*, which is a rule for mapping states S_t to actions x_t (or a_t). We use the index π to indicate the type of function, and possibly any parameters it may depend on. Thus, in an inventory problem we may use the policy to order production if $R_t < q$, in which case we order $X^\pi(R_t) = Q - R_t$. This is a (q, Q) policy (the literature refers to them as (s, S) policies) which is parameterized by q and Q. In this setting, π designates the structure of the policy ("order up to"), and the parameters (q, Q). We let Π be the

set of all policies (or all policies in a class). Later, we describe policies based on deterministic rolling horizon procedures or stochastic programming.

We assume that our policy chooses decisions based on the information available in S_t, which can only be a function of the initial state and any information that has arrived prior to time t (along with decisions that have been made prior to that time). We also assume that $x_t = X^\pi(S_t)$ is feasible with respect to any constraints at time t such as flow conservation, upper bounds and nonnegativity. We let $\mathcal{X}_t$ be the feasible region at time t, and assume that $X^\pi(S_t) \in \mathcal{X}_t$. Typically $\mathcal{X}_t$ depends on the information in S_t; we express this dependence by indexing the feasible region by time t.

Exogenous information

Exogenous information refers to random variables, which we can view as information that becomes available over time from some exogenous source (customers, weather, markets, or more mundane sources such as equipment failures). Modeling exogenous information requires the careful handling of time. Most deterministic (time dependent) models treat time as discrete, and model all events as happening at discrete points in time. When we make the transition to stochastic models, it is important to distinguish the modeling of the flow of *information* from the flow of *physical resources*.

We have found that it is useful to think of decisions as occurring in discrete time, while information arrives in continuous time. The first decision instant is $t = 0$. We denote the information arriving in continuous time using:

$$W_t = \text{New information that first becomes available between } t-1 \text{ and } t.$$

Note that our time-indexing means that W_1 represents the information arriving between $t = 0$ and $t = 1$. This also means that any variable indexed by time t is known deterministically at time t (this is not entirely standard in the stochastic optimization community). Often, we will need to represent specific random variables such as prices and demands, and for this purpose we use "hat" notation, as in $\hat{p}_t$ and $\hat{D}_t$. We would write a simple inventory problem using:

$$R_{t+1} = \max\{0, R_t + x_t - \hat{D}_{t+1}\}. \tag{10}$$

At time t, R_t is known and the decision x_t is assumed to be computed using information known at time t. The demand $\hat{D}_{t+1}$ is random at time t, but is known at time $t + 1$.

Many papers in stochastic optimization will introduce an additional bit of formalism that typically reads as follows: Let ω represent a sample path $(W_1, W_2, \ldots, W_t, \ldots)$, in which case $W_t(\omega)$ is a sample realization of information that first became available between $t - 1$ and t. Let Ω be the set of all outcomes of the entire sequence $W_1, \ldots, W_t, \ldots$, and let $\mathcal{F}$ be the sigma-algebra on Ω that defines all possible events ($\mathcal{F}$ is the set of all subsets of Ω). Also let $\mathcal{F}_t$ be the set of all events determined by the information available up to time t (sometimes written as $\sigma(W_1, \ldots, W_t)$, which is the sigma-algebra generated by $(W_1, \ldots, W_t)$). We note that

$\mathcal{F}_t \subseteq \mathcal{F}_{t+1}$, which means that $\mathcal{F}_t$ is a *filtration*. Finally, let $\mathcal{P}$ be the probability measure on $(\Omega, \mathcal{F})$. We now have a formal probability space $(\Omega, \mathcal{F}, \mathcal{P})$. When a function such as our policy depends only on information available up to t ime t, we say that X^π is $\mathcal{F}_t$-measurable. This also means that it is an *admissible policy*, *filtered policy* or that the policy is *nonanticipative*.

This type of formalism is not necessary for a proper stochastic model, but since it is widely used in the theory community, it helps readers to understand what it means.

Transition function

The transition function describes how the state variable evolves over time. We write it generically as:

$$S_{t+1} = S^M(S_t, x_t, W_{t+1}). \tag{11}$$

This function is known across communities as the state model, state equation, system model, plant model or just model. It describes the evolution of resources, information and, where appropriate, the knowledge state. The evolution of resources can usually be written as a system of linear equations, as in

$$R_{t+1} = B_t x_t + \hat{R}_{t+1}, \tag{12}$$

where $\hat{R}_{t+1}$ represents exogenous changes to inventory such as donations of blood, equipment failures and delays due to weather. Note that instead of writing constraints as we did in Eq. (6), we would break them into two sets of equations, as in

$$A_t x_t = R_t, \tag{13}$$

$$R_{t+1} - B_t x_t = \hat{R}_{t+1}. \tag{14}$$

These equations are handled very differently in a stochastic model. Equation (13) helps to define the feasible region $\mathcal{X}_t$, while (14) helps to define the transition equation.

Exogenous information often evolves independently of decisions. Thus, the price of a product may evolve according to

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

We might also have situations where the state is simply revealed, rather than being defined by some equation. So, if $\hat{D}_t$ is the demand that becomes known at time t, we could treat it as an exogenous random variable.

It is useful to think of the transition function $S^M(\cdot)$ as a composition of a resource transition function and an information transition function, which we can represent as $S^M(\cdot) = (S^R(\cdot), S^I(\cdot))$. The resource transition function might be written

$$\begin{aligned} R_{t+1} &= S^R(R_t, x_t, W_{t+1}) \\ &= B_t x_t + \hat{R}_{t+1}. \end{aligned}$$

The information transition function for this simple problem would be given by

$$p_{t+1} = S^I(I_t, x_t, W_{t+1}) = p_t + \hat{p}_{t+1}.$$

Note that the information transition function may be quite complex (this is a fairly trivial example), while the resource transition function generally has to be represented as a system of linear equations. The important idea is that the system of linear equations, quite familiar to the operations research community, is just a form of transition function, but one that is only used for the resource vector.

Objective function

We finally come to the objective function. Let

$C(S_t, x_t)$ = Thecost(ifminimizing)orcontribution(ifmaximizing)ifweareinstate S_t and make decision x_t.

It is standard in dynamic programming to explicitly write the potential dependence of the cost on the state variable, which allows us to model situations when cost parameters evolve randomly over time. Some authors will write the cost function as $C_t(S_t, x_t)$. This should only be used if the *function* depends on time, rather than depending on data (in S_t) that is a function of time.

We are now ready to write out our objective function. Note that we cannot write our objective as we did in Eq. (5), where we are optimizing over $(x_0, x_1, \ldots, x_t, \ldots)$. The problem is that x_t, for $t > 0$, is a random variable at time 0. It is precisely this distinction that makes stochastic optimization so difficult. We overcome this problem by replacing the optimization over *decisions*, x_t, with an optimization over *policies*, which we write as:

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

The objective function (15) requires a few words of explanation. First, the expectation can be thought of as an average over all the outcomes in Ω. Assume for the moment that there is a discrete set of outcomes ω, where each occurs with probability $p(\omega)$. We might generate this set of discrete outcomes through Monte Carlo simulations, where $p(\omega) = \frac{1}{N}$ if we generate N equally likely samples in Ω. In this case, we would write (15) as

$$\min_{\pi \in \Pi} \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T} C(S_t(\omega^n), X^{\pi}(S_t(\omega^n))). \tag{16}$$

Note that we have written the expectation in (15) as if it depends on the policy π. This is required if a decision might change the distribution of random events. For example, a truckload carrier will create new demands by having a truck in a region, and random travel times will increase in a network if we flow a lot of traffic over a link. If random events are not affected by decisions, then we would replace $\mathbb{E}^{\pi}$ with $\mathbb{E}$. If random events are influenced by the policy, then the sample path $(W_1(\omega), \ldots, W_t(\omega), \ldots,)$ has to be generated sequentially as decisions are made.

We make the point here that Eq. (15), combined with the transition function and a model of the exogenous information process (which defines the meaning of the expectation), is a dynamic program. Many authors equate Bellman's equation with dynamic programming. Bellman's optimality equation is a way of *solving* a dynamic program, comparable to the simplex algorithm for linear programs. This distinction is important, because it is often confused in the literature.

We note that in virtually any practical problem, the expectation in (15) cannot be computed exactly. For this reason, it is common to resort to Monte Carlo simulation, as we do in Eq. (16). For this reason, we often need to depend on noisy observations of the function

$$F^{\pi}(S_0, \omega) = \sum_{t=0}^{T} C(S_t(\omega), X^{\pi}(S_t(\omega))).$$

Finally, the most difficult challenge, of course, is to perform the search over policies π. Bellman's breakthrough was to point out that Eq. (9) characterized an optimal policy. This equation can generally be solved for problems with small, discrete state spaces, but this covers a very small class of problems in transportation (shortest path problems over networks is a nice example). It is precisely for this reason that the literature has fragmented into a range of different algorithmic strategies. While we hope to help organize this literature in the discussion below, our primary goal is to frame the discussion around the objective function in (15). For example, if there are two policies, π_1 and π_2, we can compare them by computing

$$\bar{F}^{\pi} = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T} C(S_t(\omega^n), X^{\pi}(S_t(\omega^n))),$$

for $\pi = \pi_1$ and $\pi = \pi_2$. Of course, we are limited to making statistical arguments about the performance of each policy. However, we now have a way of comparing policies that is agnostic toward any particular algorithmic strategy.

While the deterministic optimization community has had tremendous success solving linear, nonlinear and even integer programs to optimality, we suspect that finding optimal policies will remain a kind of holy grail for the vast majority of real problems in transportation and logistics. Most of the time, we will be comparing policies which hope to approximate an optimal policy in some way. This, then, is the essence of approximate dynamic programming.

Policies

Stochastic optimization problems arise in many settings, and as a result a wide range of algorithmic strategies have evolved from communities with names such as Markov decision processes, stochastic programming, stochastic search, simulation optimization, reinforcement learning, approximate dynamic programming and optimal control. A number of competing algorithms have evolved within each of

these subcommunities. Perhaps it is no wonder that stochastic optimization can seem like a jungle, lacking the elegance of a common modeling framework that has long been enjoyed by the deterministic optimization community.

Our experience across all these communities is that the vast majority of strategies for sequential stochastic optimization problems can be organized around four fundamental classes of policies:

- Myopic cost function approximations: a myopic policy is of the form

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

In some settings, we can modify the problem to get better results over time, either by modifying the cost function itself, or possibly by modifying the constraints. We can represent this using

$$X^{\text{CFA}}(S_t|\theta) = \arg\min_{x_t \in \mathcal{X}_t^\pi(\theta)} C^\pi(S_t, x_t|\theta).$$

where θ represents any tunable parameters needed to adjust the function.

- Lookahead policies: a lookahead policy optimizes over more than one time period into the future, for the purpose of making better decisions now. The most common version is to approximate the future deterministically and solve the deterministic optimization problem

$$X_t^{\text{LA-Det}}(S_t) = \arg\min_{x_t} \left(c_t x_{tt} + \sum_{t'=t+1}^{t+H} c_{t'} x_{tt'} \right), \tag{17}$$

where $\arg\min_{x_t}$ optimizes over the entire vector $x_{tt}, \ldots, x_{tt'}, \ldots, x_{t,t+H}$ over a planning horizon H, but the decision function $X_t^{\text{LA-Det}}(S_t)$ captures only x_{tt}. Of course, this has to be solved subject to constraints such as those in (6)–(8). This is referred to as a rolling horizon procedure (operations research), receding horizon procedure (computer science) or model predictive control (control theory). Since there is considerable interest in explicitly accounting for uncertainty when we make a decision, we might solve a problem of the form

$$X_t^{\text{LA-SP}}(S_t) = \arg\min_{x_t} \left(c_t x_{tt} + \sum_{\tilde{\omega} \in \tilde{\Omega}_t} p(\tilde{\omega}) \sum_{t'=t+1}^{t+H} c_{t'}(\tilde{\omega}) x_{tt'}(\tilde{\omega}) \right). \tag{18}$$

Here, $\tilde{\Omega}_t$ represents a subset of random outcomes over the interval t to $t+H$. Equation (18) is a classical two-stage stochastic programming formulation, where we first choose x_{tt}, then observe ω (which might be a sequence of random variables over time periods $t+1, \ldots, t+H$), and then choose $x_{tt'}(\omega)$. Other types of lookahead policies go under names such as tree search, decision trees and roll-out policies (Bertsekas and Castanon 1999).

- Policy function approximations: PFAs are used when the structure of the policy $X^\pi(S_t)$ (or more likely $A^\pi(S_t)$) seems obvious. A PFA is an analytic function that returns an action given a state, without solving an imbedded optimization

problem. One example is our (q, Q) inventory re-ordering policy which we can write

$$A^{\pi}(R_t) = \begin{cases} 0 & \text{If } R_t \geq q, \\ Q - R_t & \text{If } R_t < q. \end{cases} \tag{19}$$

A second example is our fleet sizing strategy. We let a_t be the decision that sets the fleet size, where we could consider using the formula

$$A^{\pi}(S_t) = \theta_0 + \theta_1(\text{ Average speed}) + \theta_2(\text{Ton-miles}).$$

This is, in fact, precisely the policy that was used at one point by a major railroad in the US. Finally, we might dispatch a shuttle bus when it has been waiting more than M minutes, or if there are more than P people on board.

- Policies based on value function approximations: VFA policies are based on Bellman's equation, and have the form

$$X^{\pi}(S_t) = \arg\min_{x_t \in \mathcal{X}_t} (C(S_t, x_t) + \mathbb{E}\{\bar{V}_{t+1}(S_{t+1})|S_t\}), \tag{20}$$

where we have used the expectation form of Bellman's equation (we replace the one-step transition matrix in (9) with an equivalent expectation). We may eliminate the expectation by using the post-decision state, giving us

$$X^{\pi}(S_t) = \arg\min_{x_t \in \mathcal{X}_t} \big(C(S_t, x_t) + \bar{V}_t^x(S_t^x)\big). \tag{21}$$

For compactness, we are writing our policies $X^{\pi}(S_t)$ as if the *function* does not depend on time (but it depends on S_t which does depend on time). There are many applications in transportation where policies are time-dependent. If the *function* depends on time, then it is appropriate to write $X_t^{\pi}(S_t)$.

Of course, it is possible to use hybrids. We might use a lookahead policy over a relatively small horizon T, and then use a value function approximation. In dynamic programming, an algorithmic strategy known as actor-critic algorithms use actions based on value function approximations to train a policy function approximation.

Of these four classes of policies, only pure lookahead policies do not use any form of functional approximation (which is part of their appeal). Functional approximations come in three basic flavors:

- Lookup tables: this requires that for each discrete state s, we have a table that specifies an action $A(s)$ or a value $\bar{V}(s)$. With lookup tables, there is a parameter (the action or the value) for each state.
- Parametric models: for a policy function approximation, this would include (q, Q) inventory policies, or our decision to adjust our fleet size based on a regression model. For value function approximations, it is very common to write these as linear models of the form

$$\bar{V}(s|\theta) = \sum_{f \in \mathcal{F}} \theta_f \phi_f(s), \tag{22}$$

where $\phi_f(s)$ is a *feature* (this can be any function of the state), $\mathcal{F}$ is the set of features and $\theta_f, f \in \mathcal{F}$ is the set of regression parameters. Neural networks have also been very popular for continuous problems that arise in engineering, but are less familiar to the transportation community. A neural network is a form of statistical model that can be thought of as a sophisticated regression equation that can be used to approximate the value of being in a state as a function of the state (see Haykin 1999 for an in-depth introduction, or Chap. 3 of Bertsekas and Tsitsiklis 1996 for a discussion of neural networks in dynamic programming).

- Nonparametric models: there is a small but growing literature proposing to use nonparametric statistics to approximate policy functions or value functions. Nonparametric methods have attracted the most attention in the context of value function approximations. Popular methods include kernel regression and support vector machines (or support vector regression).

In addition, we may use powerful hybrid strategies such as semi-parametric methods which fit parametric models around local regions of the function (see Hastie et al. 2009 for a nice overview of statistical learning methods).

We now have a fairly rich toolbox for designing approximate policies to produce practical solutions for very complex dynamic programs arising in transportation and logistics. The challenge is learning how to recognize which policy is best for a particular situation. We are going to use applications from transportation and logistics to illustration situations which are best suited for each of these policies. First, however, we are going to take a brief tour into the communities that work under the names of stochastic search and stochastic programming to demonstrate the close ties between all these communities.

Linking with other communities

Dynamic programming has long been treated as an independent field, distinct from fields such as stochastic search and stochastic programming. Careful examination reveals that these communities are all quite close, separated primarily by notation and terminology. We begin by showing the connection between stochastic search and dynamic programming, and then provide a step by step bridge from Bellman's optimality equation to the most popular tools in stochastic programming. We note that this section can be skipped without loss of continuity.

From stochastic search to dynamic programming

Problems in stochastic search are typically written

$$\min_x \mathbb{E}F(x, W), \tag{23}$$

where x is a deterministic set of parameters and W is a random variable. Stochastic search has been viewed as distinctly different from sequential decision problems (dynamic programs) because decisions x_t in sequential problems are random

variables in the future. However, this is not the right way to look at them. In a dynamic program, the optimization problem is (in most cases) a search for a deterministic *policy*, as shown in Eq. (15). We might write

$$F^{\pi}(S_0) = EF^{\pi}(S_0|W)\sum_{t=0}^{T} C(S_t, X_t^{\pi}(S_t)),$$

where $S_{t+1} = S^M(S_t, X_t^{\pi}(S_t), W_{t+1})$. The optimization problem is then

$$\min_{\pi} \mathbb{E}F^{\pi}(S_0|W). \tag{24}$$

The search for the best policy, then, is exactly analogous to the search for the best parameters x in (23).

This perspective opens up a powerful set of tools for solving dynamic programs. For example, imagine that we have a policy given by

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

While there is a substantial literature that tries to estimate θ using Bellman error minimization (so that $\bar{V}(S_t)$ predicts the value of being in a state), growing attention has been given to the idea of directly searching for the best value of θ to minimize costs. This is the same as solving Eq. (24) with $\pi = \theta$.

We can view the search over policies as a search over discrete categories of policies, as well as a search over continuous parameters (such as θ) that characterize the policy. For the vast majority of problems we are interested in, we cannot compute the expectation and instead have to depend on Monte Carlo sampling. The field of stochastic search has contributed a wide range of algorithms for dealing with these issues (see Spall 2003), and researchers in dynamic programming will come to depend on these tools.

From dynamic programming to stochastic programming

The separation of dynamic programming and stochastic programming has been created in part because of differences in problem classes, and a misunderstanding of the meaning of a dynamic program. We begin by noting that a dynamic program is a sequential (and for our purposes, stochastic) decision process. Bellman's equation (used in both dynamic programming and stochastic programming) is (a) a mathematical characterization of an optimal policy and (b) one of four potential types of policies (listed in "Policies"). Second, (multistage) stochastic programming is both a model of a sequential decision problem (i.e., a dynamic program), as well as a class of algorithmic strategies. The stochastic programming community universally uses one of two algorithmic approaches: lookahead policies [as given by Eq. (18)], and value function approximations of a lookahead model. With the latter strategy, the value function is approximated using a piecewise linear function created using Benders' cuts, although other approximation strategies are possible.

Our presentation is going to be a step-by-step tour from a classical statement of Bellman's equation (widely equated with dynamic programming), ending with two popular algorithmic strategies used in stochastic programming: lookahead policies and value function approximations. We start with discrete state and action spaces, and end with potentially high-dimensional vectors describing allocations of resources. In the process, we are going to link the standard notation of dynamic programming and stochastic programming, as well as highlight differences in perspectives.

We begin with the familiar statement of Bellman's equation (Puterman 2005)

$$V(s) = \min_{a \in \mathcal{A}} \left(C(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V(s') \right).$$

This formulation applies to stationary, infinite horizon problems, and generally assumes that states s and actions a are discrete. Most problems in transportation are better modeled as time-dependent problems, which would be written

$$V_t(S_t) = \min_{a_t \in \mathcal{A}_t} \left(C(S_t, a_t) + \gamma \sum_{s' \in \mathcal{S}} P(S_{t+1} = s'|S_t, a_t) V_{t+1}(s') \right). \tag{25}$$

where $S_{t+1} = S^M(S_t, a_t, W_{t+1})$. A more natural form is to replace the one-step transition matrix (which we could never possibly compute) with an expectation

$$V_t(S_t) = \min_{a_t \in \mathcal{A}_t} (C(S_t, a_t) + \gamma \mathbb{E}\{V_{t+1}(S_{t+1})|S_t\}). \tag{26}$$

Here, we can interpret the expectation as a sum (or integral) over the random variable W_{t+1}, but we can also interpret it as a conditional expectation over the entire set $\omega \in \Omega$ representing observations of $W_1, W_2, \ldots, W_T$ (this is the perspective of the probability community). In this case, to be accurate we need to also index the expectation by the policy π since this influences the event that we are in a particular state S_t. We can avoid this by viewing the expectation as a sum over all outcomes of $W_{t+1}, \ldots, W_T$, given that we start in state S_t. There is an implicit assumption that we are using an optimal policy from time $t+1$ onward.

Of course, we cannot compute the expectation for most problems in practice. We now use the fact that the value function is the sum of all future costs. We are going to briefly assume that we can fix the policy represented by $A_t^\pi(S_t)$ (we are going to assume that our policy is time-dependent), and we are going to avoid for the moment the issue of how to compute this policy. This gives us

$$V_t^\pi(S_t) = \min_{a_t \in \mathcal{A}_t} \left(C(S_t, a_t) + \mathbb{E}\left\{ \sum_{t'=t+1}^{T} \gamma^{t'-t} C(S_{t'}, A_{t'}^\pi(S_{t'})) \middle| S_t \right\} \right). \tag{27}$$

We can not compute the expectation, so we approximate it by replacing it with an average over a set of sample paths given by the set $\hat{\Omega}$. We also use this opportunity to make the transition from scalar actions a to vector decisions x, which is more useful in transportation and logistics. For the moment, we will assume x_t is given by

a yet-to-be determined function (policy) $X^\pi(S_t)$ which returns a feasible vector $x_t \in \mathcal{X}_t$. This gives us

$$\bar{V}_t^\pi(S_t) = \min_{x_t \in \mathcal{X}_t} \left(C(S_t, x_t) + \frac{1}{|\hat{\Omega}|} \sum_{\omega \in \hat{\Omega}} \sum_{t'=t+1}^{T} \gamma^{t'-t} C(S_{t'}(\omega), X_{t'}^\pi(S_{t'}(\omega))) \right). \tag{28}$$

We have replaced $V_t^\pi(S_t)$ with $\bar{V}_t^\pi(S_t)$ to indicate that we are now computing a statistical estimate. We can compute $\bar{V}_t^\pi(S_t)$ by simulating our policy $X_{t'}^\pi(S_{t'})$ over time periods $t+1, \ldots, T$ for each sample path ω.

At this point we have specified a model and have an expression for the approximate value of a policy, but we have not touched on the problem of how to compute the policy. The stochastic programming community uses two classes of policies: a stochastic lookahead policy which explicitly optimizes over a horizon $t, \ldots, t+H$, and a policy based on a value function approximation (but still limited to a lookahead horizon). These are described below.

Stochastic programming as a lookahead policy

We start by creating what we will call a *lookahead model* by constructing $\hat{\Omega}$ in a particular way. We first point out that the state $S_{t'}$ consists of two components: the resource state $R_{t'}$ that is controlled by our decisions $x_t, \ldots, x_{t'-1}$, and the exogenous information that we have been calling $I_{t'}$. In stochastic programming, it is common practice to view the exogenous information state as the entire history. This means that our history up to time t' consists of

$$h_{tt'} = (W_{t+1}, W_{t+2}, \ldots, W_{t'}).$$

Our state at time t' (given that we are starting at time t) is then given by $S_{t'} = (R_{t'}, h_{tt'})$. Technically we should index the state as $S_{tt'}$, but we are going to make the indexing on time t implicit.

It is important to generate the samples in $\hat{\Omega}$ in the form of a scenario tree. Imagine that we have generated a specific history $h_{t'}$ (dropping the index t). This is called a node in the scenario tree, and from this node (this history), we can generate multiple sample paths by sampling multiple realizations of $W_{t'+1}$ that may depend on the history $h_{t'}$. Each of these histories map to a specific outcome $\omega \in \hat{\Omega}$, but there may be multiple outcomes that match a particular history up to time t'. Let $\mathcal{H}_{t'}(h_{t'})$ be the set of outcomes $\omega \in \hat{\Omega}$ where the partial sequence $(W_{t+1}(\omega), W_{t+2}(\omega), \ldots, W_{t'}(\omega))$ matches $h_{t'}$.

We are going to make decisions $x_{t'}$ that depend on the information available at time t'. Imagine that we are at a node in the scenario tree corresponding to a particular history $h_{t'}$. To get to time t', we would have had to have made a sequence of decisions $x_t, \ldots, x_{t'-1}$. This history of decisions, combined with the new information $\hat{R}_t$, brings us to the new resource variable $R_{t'}$. Thus, our state variable is now given by $S_{t'} = (R_{t'}, h_{t'}) = (x_{t'-1}, h_{t'})$ (we assume that $R_{t'}$ is completely determined by $x_{t'-1}$ and $h_{t'}$).

We now have the notational machinery to solve Eq. (28) for the optimal policy for our lookahead model. We are going to use the fact that the resource state variables $R_{t'}$ are all linked by a set of linear equations. We do not care how the information state variables $h_{t'}$ are linked as long as we can enumerate all the histories *independent of prior decisions*. However, it is important to recognize that the information variables (the histories) *are* linked through a transition function which is implicit in the process for generating scenario trees. This allows us to optimize over all the decisions x_t and $x_{t'}(\omega), t' > t, \omega \in \Omega$ as one large linear (and possibly integer) program:

$$\bar{V}_t(S_t) = \min_{x_t} \left(C(S_t, x_t) + \frac{1}{|\hat{\Omega}|} \sum_{\omega \in \hat{\Omega}} \min_{x_{t+1}(\omega), \ldots, x_T(\omega)} \sum_{t'=t+1}^{T} \gamma^{t'-t} C(S_{t'}(\omega), x_{t'}(h_{t'}(\omega))) \right). \tag{29}$$

The key insight in stochastic programming (when we are using a lookahead policy) is recognizing that if we generate the information states in the form of the histories $h_{t'}(\omega)$ in advance, the decisions $x_{t'}(h_{t'})$ are linked through the resource transition function $R_{t'} = B_{t'-1}x_{t'-1} + \hat{R}_{t'}$, and therefore are implicitly a function of $R_{t'}$ (which means we could have written the decisions as $x_{t'}(S_{t'})$). As a result, Eq. (29) can be solved as a single large, deterministic linear (or integer) program. This allows us to avoid designing an explicit policy $X^{\pi}_{t'}(S_{t'})$. This is conceptually the same as solving a decision tree (with discrete actions a), but can be used when decisions are vectors.

We have to recognize that $\hat{\Omega}$ is the set of all histories that we have sampled, where an element $\omega \in \hat{\Omega}$ refers to an entire history from t to T. Then, we can interpret $x_{t'}(\omega)$ as $x_{t'}(h_{t'}(\omega))$ as the decision that depends on the history $h_{t'}$ produced by the sample path ω. To illustrate, imagine that we are solving a problem for time periods t, $t+1$ and $t+2$, where we generate 10 observations of W_{t+1} and, for each of these, 10 more outcomes for W_{t+2}. This means that we have 100 elements in $\hat{\Omega}$. However, we have to choose a single vector x_t, 10 vectors x_{t+1} and 100 vectors x_{t+2}.

An alternative way of interpreting Eq. (29), however, is to write $x_{t'}(\omega)$ explicitly as a function of ω, without inserting the interpretation that it only depends on the history up to time t'. This interpretation is notationally cleaner, and also represents what the field of stochastic programming actually does. However, it then introduces an important complication. If we index $x_{t'}(\omega)$ on ω, then this is the same as indexing it on the entire history from t to T, which means we are allowing the decision to see into the future.

To handle this, we first make one last modification to our objective function by writing it in the form

$$\bar{V}_t(S_t) = \frac{1}{|\hat{\Omega}|} \sum_{\omega \in \hat{\Omega}} \min_{x_t(\omega), \ldots, x_T(\omega)} \sum_{t'=t}^{T} \gamma^{t'-t} C(S_{t'}(\omega), x_{t'}(h_{t'}(\omega))). \tag{30}$$

We need to solve this optimization subject to the constraints for $t' > t$ and all $\omega \in \hat{\Omega}$,

$$A_t x_t(\omega) = b_t, \tag{31}$$

$$x_t(\omega) \geq 0, \tag{32}$$

$$A_{t'}(\omega)x_{t'}(\omega) - B_{t'-1}(\omega)x_{t'-1}(\omega) = b_{t'}(\omega), \tag{33}$$

$$x_{t'}(\Omega) \geq 0. \tag{34}$$

All we have done here is to include x_t (the "here and now" decision) in the summation over all the scenarios Ω. Returning to our little example with three time periods, we now have three sets of decisions x_t, x_{t+1} and x_{t+2} with 100 outcomes, creating 300 vectors that we have to choose. If we solved this problem without any further changes, it would be like solving 100 deterministic problems, where each set of vectors (x_t, x_{t+1}, x_{t+2}) depends on ω. To implement this solution we have to know the entire future before choosing x_t. This is an example of an *inadmissible* policy (alternatively, we might say that the solution is *anticipative* because it is allowed to anticipate the future).

The stochastic programming community fixes this problem in the following way. We do not want to have two different decisions for $x_{t'}$ for two different outcomes ω that share the same history up to time t'. So we can introduce the constraint

$$x_{t'}(\omega) - \bar{x}_{t'}(h_{t'}) = 0, \quad \forall \omega \in \mathcal{H}_{t'}(h_{t'}). \tag{35}$$

Here, $\bar{x}_{t'}(h_{t'})$ is a new set of variables where there is one variable for each history, subject to constraints that depend on $R_{t'}$ which depends on the prior decisions. Equation (35) is known as a *nonanticipativity constraint* because it requires each $x_{t'}(\omega)$ for all $\omega \in \mathcal{H}_{t'}(h_{t'})$ (that is, the set of all outcomes that share a common history) to be the same. Note that if $t = t'$, there is only a single history h_t (determined by the state S_t that we are in at time t).

Equation (30), then, is optimizing over all policies in the set of lookup table policies by choosing the best vector $x_{t'}$ for each history $h_{t'}$. The key feature (when dealing with vector-valued decisions) is that the optimization problem defined by (30)–(34), along with the nonanticipativity Eq. (35), is a linear program (integrality constraints may also be included). The only problem is that for many applications, it is an extremely large linear (or integer) program. For this reason, it is common to simplify the scenario tree by limiting the number of times that information can be revealed. In fact, a popular strategy is to assume that after making the initial decision x_t, the information over the entire remainder of the horizon is revealed, which means that the vector x_{t+1} is allowed to see $W_{t'}$ for $t' > t + 1$. This is a form of cheating (peeking into the future), but we tolerate it because we are not actually going to implement the decisions $x_{t'}$ for $t' > t$.

Instead of using Eq. (28) as the value of a policy, we can modify it slightly to take the form of a lookahead policy with the form

$$X_t^{\mathrm{LA-SP}}(S_t) = \arg \min_{x_t(\omega),\ldots,x_T(\omega),\omega\in\hat{\Omega}} \sum_{t'=t}^{t+H} \gamma^{t'-t} C(S_{t'}(\omega), x_{t'}(h_{t'}(\omega))). \tag{36}$$

subject to (31)–(34) and the nonanticipativity constraints (35). We have written our lookahead policy as spanning a planning horizon from t to $t + H$ instead of T since there are many problems where T may be hundreds or thousands of time periods

into the future, but where we can obtain good decisions by planning over a shorter horizon. We assume that the arg min in (36) returns only x_t.

In practice, computing this policy can be computationally very demanding. It is because of this computational challenge that a substantial community has evolved to solve this problem (see, for example, Rockafellar and Wets 1991; Birge and Louveaux 1997; Romisch and Heitsch 2009). However, it is also important to recognize that even an optimal solution to (36) does not constitute an optimal policy, in part because of the truncated planning horizon H and also because of the need to use a sampled scenario tree. Solving the lookahead model optimally does not translate to an optimal policy of the original problem.

Stochastic programming using a value function approximation

As a result of the computational demands of solving lookahead models directly, the stochastic programming community has long recognized that there there is an alternative strategy based on approximations of value functions (known as recourse functions in stochastic programming). Using the notational system in Shapiro et al. (2009), the strategy starts by writing

$$Q_t(x_{t-1}, \xi_{[t]}) = \min_{x_t} \big(c_t x_t + \mathbb{E}\{Q_{t+1}(x_t, \xi_{[t+1]})|\xi_{[t]}\}\big). \tag{37}$$

Now we just have to translate the notation back to ours. Q_t is called the recourse function, but this is just different terminology and notation for our value function V_t. $\xi_{[t]}$ is the history of the exogenous information process up to time t (which we refer to as h_t). The resource vector R_t is a function of x_{t-1} and, if we have an exogenous component such as $\hat{R}_t$ [as in Eq. (12)], then it also depends on W_t (which is contained in $\xi_{[t]}$). This means that the state variable is given by $S_t = (R_t, h_t) = (x_{t-1}, \xi_{[t]})$. We note that it is mathematically equivalent to use x_{t-1} instead of R_t, but in most applications R_t is lower dimensional than x_{t-1} and would be more effective computationally as a state variable.

We still face the challenge of optimizing (over a vector-valued decision x_t) the imbedded expectation of the unknown function $Q_{t+1}(x_t, \xi_{[t+1]})$. It is possible to show that (37) can be replaced with

$$Q_t(x_{t-1}, \xi_{[t]}) = \min_{x_t \in \mathcal{X}_t, v} (c_t x_t + v), \tag{38}$$

where

$$v \geq \alpha_t^k(\xi_{[t]}) + \beta_t^k(\xi_{[t]})x_t, \quad \text{for } k = 1, \ldots, K, \tag{39}$$

and where $\mathcal{X}_t$ captures the feasible region for x_t [such as Eqs. (31)–(32)]. Here, Eq. (39) is generated by solving the dual problem for time $t + 1$, which means that K depends on the number of iterations that have been executed. We note that in the stochastic programming community, Eq. (39) would be written

$$v \geq \alpha_{t+1}^k(\xi_{[t]}) + \beta_{t+1}^k(\xi_{[t]})x_t, \quad \text{for } k = 1, \ldots, K, \tag{40}$$

where α and β are now indexed by $t+1$ instead of t. The style in Eq. (39) is consistent with our indexing where any variable indexed by t can be computed at time t. The style in Eq. (40) is used to communicate that the cuts are approximating the recourse function at time $t+1$, although it is really more accurate to say that it is approximating the recourse function around the post-decision state at time t given by $\mathbb{E}\{Q_{t+1}(x_t, \xi_{[t+1]})|\xi_{[t]}\} = V_t^x(S_t^x) = V_t^x(R_t^x, h_t) = V_t^x(x_{t-1}, h_t)$.

A rich history has grown within the stochastic programming community around the use of Benders' cuts. Important references include Higle and Sen (1991), Kall and Wallace (1994), Higle and Sen (1996), Birge and Louveaux (1997), and Shapiro et al. (2009). This literature has established convergence results that show that the solution converges to the optimal solution of the lookahead model (not to be confused with the optimal policy of the original problem).

The "cuts" represent a form of nonparametric value function approximation that takes advantage of the convexity of the value function. The vector x_t in Eq. (39) can be replaced by R_t^x, the post-decision resource vector (sometimes dramatically reducing its dimensionality). These cuts are also indexed by $h_t = \xi_{[t]}$, which means that our value function approximation depends on the full post-decision state vector $S_t^x = (R_t^x, h_t) = (R_t^x, I_t)$, which consists of the resource vector after we make a decision, and the current (but not the future) information state.

Using this value function approximation, we can write the policy as

$$X_t^\pi(S_t) = \arg \min_{x_t \in \mathcal{X}_t, v} (c_t x_t + v), \tag{41}$$

subject to Eq. (39). Here, the index π refers not only to the structure of the policy, but also the cuts themselves.

Benders' cuts have attracted considerable attention in the operations research community, although there appear to be rate of convergence issues when the dimensionality of the resource variable R_t grows (Powell et al. 2004). Another approximation strategy is a value function that is linear in the resource variable, as in

$$\bar{V}_t(S_t) = \min_{x_t \in \mathcal{X}_t} \left(c_t x_t + \sum_i \bar{v}_{ti} R_{ti}^x \right), \tag{42}$$

where R_t^x is the post-decision resource vector produced by x_t (this can be written in the general form $R_t^x = B_t x_t$). A third approximation is a value function that is piecewise linear but separable in R_t^x, as in

$$V_t(S_t) = \min_{x_t \in \mathcal{X}_t} \left(c_t x_t + \sum_i \bar{V}_{ti}(R_{ti}^x) \right). \tag{43}$$

Note that the approximations $\bar{v}_{ti}$ and $\bar{V}_{ti}(R_{ti})$ are not indexed by the history h_t (although we could introduce this), making these methods computationally much more compact, but with a loss in optimality proofs (which we would lose anyway as a result of the use of linear or piecewise linear, separable approximations). However, these techniques have proven useful in industrial problems with hundreds or

thousands of time periods (stages), which could never be tackled using scenario trees (see Simao et al. 2009, Topaloglu and Powell 2006 for illustrations).

There is a critical difference in how the stochastic programming community uses value function approximations [such as Eqs. (38)–(39)] versus the strategies used by the approximation dynamic programming community [such as Eqs. (42) or (43)]. The issue is not the use of Benders' cuts vs linear or piecewise linear, separable approximations. The difference is the dependence on scenario trees. By indexing the cuts on the scenario tree, the stochastic programming community is designing an algorithm that optimizes the lookahead model rather than the full model. By contrast, the ADP community will train their value functions on the full model. The SP community hopes that near-optimal solutions to the lookahead model (defined in terms of the scenario tree) will produce good solutions for the full model. The ADP community can, in special cases, obtain optimal policies to the full model, but for most applications limitations in machine learning limit the accuracy of the value function approximations.

Shortest path problems

Imagine that we have a graph where we have to get from an origin o to a destination d over a graph where the cost to traverse arc (i,j) is given by c_{ij}. It is well known that we can find a shortest path using Bellman's optimality equation

$$V_t(i) = \min_{a \in \mathcal{A}_i}(c_{ia} + V_{t+1}(a)),$$

where $\mathcal{A}_i$ is the set of nodes j that are incident to node i, and the action a is one of these nodes. What makes dynamic programming work so well in this setting is that the state space, the set of nodes, is quite manageable.

Imagine now that the costs c_{ij} are stochastic where, prior to traversing a link, we know only the distribution of the costs. Let $\hat{c}_{ij}$ be the random variable representing the actual cost of traversing (i,j) with expectation $c_{ij} = \mathbb{E}\hat{c}_{ij}$. This problem is no more complicated than our original deterministic problem, since we would plan our path using the expectations.

Finally, let's imagine that when we arrive at node i, we get to see the realization of the costs $\hat{c}_{ij}$ on links out of node i, where the costs may be continuous. This problem is suddenly quite different. First, our state variable has changed from the node where we are located, to the node along with the observed costs. Let R_t be the node where we are located after t transitions (this is the state of our resource), and let $I_t = (\hat{c}_{r,j})_{j \in \mathcal{A}_r}$ be the costs on links incident to the node $r = R_t$ (we think of this as information we need to make a decision). Our state variable is then given by $S_t = (R_t, I_t)$. When we think of the size of our state space, this problem is dramatically more complex.

We can overcome this complication using our post-decision state variable. If we are at node i and decide to go to node j, our post-decision state is the state immediately after we have made this decision (which is to say, our intended

destination j), but before we actually traverse to node j. This means that we have not observed the costs on arcs incident to node j. As a result, we can think of the post-decision state S_t^a at time t as the node where we would be at time $t+1$ but without the information that would be revealed at time $t+1$.

Assume that we are traversing the path repeatedly. On iteration n at time t, we are in state $S_t^n = (R_t^n, I_t^n)$. Let $r = R_t^n$ and let the costs $\hat{c}_{ra}^n$ be contained in I_t^n. Using the post-decision state variable, we can write Bellman's equations using

$$V_t(S_t^n) = \min_{a \in \mathcal{A}_r} \left(\hat{c}_{ra}^n + V_t^a(S_t^a) \right), \tag{44}$$

$$V_t^a(S_t) = \mathbb{E}\{V_{t+1}(S_{t+1}) | S_t\}, \tag{45}$$

where the post-decision state is given by $S_t^a = a$. Equation (44) is deterministic, and the post-decision state space for S_t^a is small, making (44) relatively easy to compute. The problem is that we may have trouble finding the expectation in Eq. (45). We can get around this by replacing $V_t^a(s)$ with a statistical approximation. Imagine that we are at a particular state S_t^n where $R_t^n = i$ and where $\hat{c}_{ij}^n$ for $j \in \mathcal{A}_i$ are known. Let

$$\hat{v}_t^n(i) = \min_{j \in \mathcal{A}_i} \left(\hat{c}_{ij}^n + \bar{V}_{t+1}^{n-1}(j) \right) \tag{46}$$

be a sample realization of the value of being in state S_t^n, which means we are at node i at time t, observing the costs $\hat{c}_{ij}^n$. We could also say that this is a sample realization of the value of being in the pre-decision state $S_{t-1}^a = i$ (at time $t-1$, we have made the decision to go to node i, but have not yet arrived at node i). We now use this to update our estimate of the value of being in the *previous post-decision state* S_{t-1}^a using

$$\bar{V}_{t-1}^n(S_{t-1}^{a,n}) = (1 - \alpha_{n-1}) \bar{V}_{t-1}^{n-1}(S_{t-1}^{a,n}) + \alpha_{n-1} \hat{v}_t^n(i), \tag{47}$$

where $i = R_t^n$ and $0 < \alpha_{n-1} \leq 1$ is a stepsize.

An alternative strategy would be to use the actions from solving (46) to produce an entire sequence of actions (say, out to an end of horizon T). Let the sequence of states generated by the actions from this policy be denoted by $(S_0^n, a_0^n, S_1^n, a_1^n, \ldots, S_T^N)$. Now, compute $\hat{v}_t^n(i)$ using

$$\hat{v}_t^n = \sum_{t'=t}^{T} C(S_{t'}^n, a_{t'}^n) + \bar{V}_{T+1}^n(S_{T+1}). \tag{48}$$

This is accomplished using a simple backward traversal. We might assume that $\bar{V}_{T+1}^n(S_{T+1}) = 0$, but we include this last term just in case.

These ideas here are presented using the setting of discrete action spaces, which are useful if we are routing, for example, a single vehicle around a network. For problems involving, say, fleets of vehicles, we can use similar ideas if we think of $\hat{v}_t^n$ as the *marginal* value of additional resources (see Simao et al. 2009 for an illustration in the setting of managing fleets of drivers).

This simple example has allowed us to provide our first illustration of approximate dynamic programming based on a value function approximation (using a lookup table), taking advantage of the post-decision state. Unfortunately,

we have not yet created an algorithm that computes the best path. One strategy is to loop over all the nodes in the network, computing (46) and then (47) to update the approximate values at every state. For deterministic problems, we would say that such an algorithm is hopelessly inefficient but at least it would work. When we depend on Monte Carlo samples of costs, such an algorithm is not guaranteed to provide the optimal path, even in the limit.

There is a slightly different algorithm which is computationally even clumsier, but at least we get a convergence proof. It has been developed in the reinforcement learning community under the name "Q-learning" and it proceeds by iteratively estimating Q-factors which are estimates of the value of being in a state s (that is, sitting at a node) *and* taking an action a (corresponding to a link). Assume we choose at iteration n a state-action pair (s^n, a^n). The updates are computed using

$$\hat{q}^n(s^n, a^n) = C(s^n, a^n) + \gamma \min_{a'} \bar{Q}^{n-1}(s^{n+1}, a'), \tag{49}$$

$$\bar{Q}^n(s^n, a^n) = (1 - \alpha_{n-1})\bar{Q}^{n-1}(s^n, a^n) + \alpha_{n-1}\hat{q}^n(s^n, a^n). \tag{50}$$

Here, s^{n+1} is a downstream state that is randomly sampled given the current state s^n and action a^n. Note that s^n must be a pre-decision state, so for our stochastic shortest path application, it would have to include the vector of costs. The algorithm is provably convergent (Tsitsiklis 1994), but it is also known to be impractically slow.

The algorithm we would like to use (and which we will use later), works as follows. Imagine that we could start with an initial set of approximations $\bar{V}^0(i)$ for each node i, and then use the action selected in Eq. (46) to determine the next link to traverse. We can use this idea to simulate our way through the network, a process that is practical regardless of the complexity of the state variable. This process would then have to be repeated a number of times, with the hope that we would learn the shortest path.

If we could compute the expectation exactly, and if we start with optimistic estimates of the value of being at each node, this algorithm actually works [in its deterministic form, this is called the A* algorithm (Pearl 1984), and was later re-introduced under the name "real-time dynamic programming" (Barto et al. 1995)]. But when we depend on Monte Carlo sampling, this simple, elegant strategy fails in this setting. The reason is widely known as the "exploration versus exploitation" problem (see Powell 2011 [Chap. 12]; Powell and Ryzhov 2012). We do not learn about a state unless we visit it. If our estimate of the cost generated from a state is too high, we might not visit it and would therefore never realize that the cost is incorrect. For problems with small action spaces, it is common to introduce an exploration policy where we might choose an action just to try it. Even with small action spaces, such algorithms will work only when there is some form of generalized learning, where visiting one state teaches us something about other states (we have an example of this in "Fleet management problems" for the setting of a single truck). For vector-valued decisions, these exploration policies would never work.

However, there are problems where this strategy works beautifully, one being inventory problems which we review next. It is going to be important for us to understand when this powerful idea works, and when it does not.

Inventory problems

Inventory problems are fundamental to operational planning in transportation and logistics, ranging from classical applications in retail management to decisions about the size of trailer pools at truck terminals or the number of locomotives or jets that railroads or airlines should own. Shipping companies need to solve inventory problems to determine how many containers to keep at a port or to plan advance purchases of capacity on container ships. Inventory problems are also at the heart of fleet management problems faced in trucking, rail and shipping.

A basic model

We use this problem to illustrate the five components of a model of a dynamic program. Assume that we are managing the trailer pool of containers at a port. We receive a revenue p_t for serving demands D_t at time t. We can order new trailers moved into the terminal for a cost c_t. In addition, the trailer pool changes randomly due to the need to move loads out of the port, or from trailers pulling loads into the port. Unsatisfied demands are lost. We model this problem as follows:

- State variable: let R_t be the number of trailers on hand at the end of day t. Assume that the price p_{t+1} depends on the price p_t, but that the costs c_t are independent and identically distributed, as are the demands D_t. Our state variable is then $S_t = (R_t, D_t, p_t, c_t)$.
- Decision variable: let x_t be the number of trailers that we request to be moved empty into ($x_t > 0$) or out of ($x_t < 0$) the yard, at a unit cost c_t.
- Exogenous information: let $\hat{p}_t$ be the change in the price between $t-1$ and t (that becomes known at time t), and let $\hat{D}_t$ be the demands that become learned at time t (equivalently, between $t-1$ and t) to be served at time t or later. Also assume that $\hat{c}_t$ is revealed at time t. Finally, let $\hat{R}_t$ capture changes in the trailer inventory due to the movement of loaded trailers into the port ($\hat{R}_t > 0$) or out of the port ($\hat{R}_t < 0$). Our exogenous information process is given by $W_t = (\hat{D}_t, \hat{R}_t, \hat{p}_t, \hat{c}_t)$. Finally, we note that the costs $\hat{c}_t$ increase when we need to order more trailers, since we have to get them from more distant locations, which means that the random variables depend on our decisions (and hence the policy).
- Transition function: our state variables evolve according to

$$
\begin{aligned}
R_{t+1} &= R_t + x_t + \hat{R}_{t+1},\\
D_{t+1} &= \hat{D}_{t+1},\\
p_{t+1} &= p_t + \hat{p}_{t+1},\\
c_{t+1} &= \hat{c}_{t+1}.
\end{aligned}
$$

Note that D_{t+1} and c_{t+1} do not depend on history, so we assume that these are simply revealed. We observe the change in prices (rather than just the new price) so that we can demonstrate the dependence on history (the random variable $\hat{p}_{t+1}$ may

depend on p_t). Finally, R_{t+1} is governed by both a decision as well as exogenous inputs.

- Objective function: let $X^{\pi}(S_t)$ represent our decision function (policy), and let $C(S_t, x_t)$ be the cost incurred during time period t. Our objective function is then given by

$$\min_{\pi} \mathbb{E}^{\pi} \sum_{t=0}^{T} \gamma^t C(S_t, X^{\pi}(S_t)).$$

We note that while the pre-decision state variable S_t has four dimensions, the post-decision state $S_t^x = (R_t^x, p_t)$, where $R_t^x = R_t + x_t$, has only two dimensions.

Solution based on Bellman's equation

Inventory problems represent an ideal setting for illustrating the power of Bellman's optimality equation. Assume that we discretize $S_t = (R_t, D_t, p_t, c_t)$ into a reasonable number of potential states. We can use Bellman's equation to characterize an optimal policy, given by

$$V(S_t) = \min_{x \in \mathcal{X}} (C(S_t, x) + \gamma \mathbb{E}\{V(S_{t+1}) | S_t\}). \tag{51}$$

where $S_{t+1} = S^M(S_t, x, W_{t+1})$ and the expectation is over the random variable W_{t+1}. Interestingly, the dynamic programming community has focused far more attention on the infinite horizon problem. A classical strategy is an algorithm known as value iteration, which starts with an estimate $V^0(s)$ for the value of being in each state $s \in \mathcal{S}$, and then produces updates using

$$V^n(s) = \min_{x \in \mathcal{X}} \big(C(s, x) + \gamma \mathbb{E}\{V^{n-1}(s') | s\}\big), \tag{52}$$

where $V^n(s)$ is computed for each state $s \in \mathcal{S}$. This strategy is also known as exact value iteration, in contrast with the update described in Eqs. (46)–(47), which is known as approximate value iteration. We note that exact value iteration enjoys a rigorous convergence theory, while approximate value iteration only works with special structure. Fortunately, convexity happens to be a powerful property that allows approximate value iteration to work.

Using this strategy, we can state our policy using the value function as

$$X^{\pi}(s) = \arg\min_{x \in \mathcal{X}} (C(s, x) + \gamma \mathbb{E}\{V(s') | s\}), \tag{53}$$

which is a function that we can execute on the fly [i.e., given a state s and value function $V(s')$, we can solve (53) to obtain an action]. Alternatively, we can execute (53) for each state s, determine the action and store it in a matrix that we can call $X^{\pi}(s)$, where π is the optimal policy. While these policies are mathematically equivalent, they are computationally quite different. We would refer to the policy in (53) as a policy based on a value function approximation (even with exact value

iteration, the value function is never truly exact), while storing the matrix $X^{\pi}(s)$ is a policy function approximation using a lookup table representation.

Policy search

We feel that we can use a policy such as a rule that we place an order when the inventory R_t is less than a parameter q, and we order enough to take the inventory up to Q. This is a form of policy function approximation, using a parametric policy. We can find the best values for q and Q by solving

$$\min_{(q,Q)} F(q,Q) = \mathbb{E}^{\pi} \sum_{t=0}^{T} C(S_t, X^{\pi}(S_t|q,Q)). \tag{54}$$

This problem can be solved using the techniques of stochastic search (see Spall 2003 for a thorough review, and Powell 2011 [Chap. 7] for stochastic search in the context of dynamic programs).

In practice, it is often the case that problems are nonstationary, reflecting time-of-day or day-of-week patterns. If this is the case, we might want to use a vector (q_t, Q_t), which means we now have two parameters for each time period. Such a problem is much harder, especially if we do not have access to derivative information. Interestingly, value function approximations can be adapted to handle time dependent problems quite easily. In Powell (2011), value function approximations are used to solve an inventory problem with 175,000 time periods.

Exploiting convexity

Another strategy is to develop a value function approximation that exploits the fact that for many (although not all) inventory problems, the value function is convex in the inventory R_t. Assume that we are going to represent the value function using a piecewise linear approximation, where $\bar{v}^n(r)$ is an estimate of the slope of $V(R)$ after n observations for $R = r$. Our optimization problem at iteration n would then look like

$$\tilde{V}_t^n(R_t) = \min_{x,y} \left(C(R_t^n, x) + \gamma \sum_{r=0}^{R^{\max}} \bar{v}_t^{n-1}(r) y_r \right), \tag{55}$$

where $\sum_{r=0}^{R^{\max}} y_r = R_t + x = R_t^x$ is the amount of inventory after our decision (the post-decision state variable). Now let

$$\hat{v}_t^n = \tilde{V}_t^n(R_t^n + 1) - \tilde{V}_t^n(R_t^n) \tag{56}$$

be an estimate of the slope of the value function when $R_t = R_t^n$. We can use this to update our estimate of the slope at $r = R_t^n$ for the value function approximation at the previous, post-decision state variable $R_{t-1}^{x,n}$ using

$$\bar{v}_{t-1}^n(R_{t-1}^{x,n}) = (1 - \alpha_{n-1}) \bar{v}_{t-1}^{n-1}(R_{t-1}^{x,n}) + \alpha_{n-1} \hat{v}_t^n. \tag{57}$$

We have one last hurdle. If the value function is convex, then we would have that

$\bar{v}_t^n(r+1) \geq \bar{v}_t^n(r)$. While this may be true for $\bar{v}_t^{n-1}(r)$, it may no longer be true after the update from Eq. (57), because of the noise in $\hat{v}_t^n$. If we lose convexity, there are a number of ways to restore it (Powell and Godfrey 2001; Powell et al. 2004; Powell 2011 [Chap. 13]). Maintaining convexity accelerates convergence, and later we will exploit it when our actions become vectors.

This problem can be solved using approximate value iteration, as depicted in Fig. 1. Here, we use our approximation of the value function to choose an action, and we use information from this decision problem, in the form of the slope $\hat{v}_t^n$, to update the slope of the value function around the previous, post-decision state variable.

Now we have to pause and observe: This is exactly the algorithm that we wanted to use for the stochastic shortest path problem, but we claimed it would not work. Here, we claim it does work. Furthermore, it can be proven to produce an asymptotically optimal solution, and it works very well in practice (with a caveat). Powell and Simao (2009) use this strategy in the planning of high-value spare parts for a manufacturer of regional aircraft. This work considered hundreds of spare parts, including parts that ranged from demands of several per day to truly high value spare parts (wings, engines) which were only used for instances of extremely rare failures. Such parts might be required once every two or three years. Since the system had to be modeled in daily time steps (to model parts with much higher demands), the result was a system that exhibited hundreds of time periods between demands.

Approximate value iteration is especially practical for problems in transportation and logistics, since it is basically a simulator, which scales very well in industrial applications. However, we have to be aware that we are caught in this tension between problems where it works extremely well (such as our inventory problem) and where it does not work at all (our stochastic shortest path problem). Perhaps it is not surprising that convexity is the critical property. We have had success with this

Step 0. Initialization:

Step 0a. Initialize an approximation for the value function $\bar{V}_t^0(S_t^a)$ for all post-decision states S_t^a, $t = \{0, 1, \ldots, T\}$.

Step 0b. Set $n = 1$.

Step 0c. Initialize $S_0^{a,1}$.

Step 1. Choose a sample path ω^n.

Step 2. Do for $t = 0, 1, \ldots, T$:

Step 2a: Let

$$x_t^n = \arg\min_{x \in \mathcal{X}} \left(C(R_t^n, x) + \gamma \sum_{r=0}^{R^{max}} \bar{v}_t^{n-1}(r) y_r \right) \tag{58}$$

where $\sum_{r=0}^{R^{max}} y_r = R_t + x = R_t^x$. Compute $\hat{v}_t^n$ using

$$\hat{v}_t^n = \bar{V}_t^n(R_t^n + 1) - \bar{V}_t^n(R_t^n)$$

Step 2b. Use $\hat{v}_t^n$ to update $\bar{V}_{t-1}^n$ while maintaining convexity.

Step 2c. Sample $D_{t+1}^n = D_{t+1}(\omega^n)$ and compute the next state using

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

Step 3. Increment n. If $n \leq N$ go to Step 1.

Step 4. Return the value functions $(\bar{V}_t^n)_{t=1}^T$.

Fig. 1 Approximate value iteration for an inventory problem

algorithmic strategy in a number of fleet management problems (Godfrey and Powell 2002; Topaloglu and Powell 2006; Powell and Topaloglu 2005; Powell et al. 2012a, b), but all of these are convex.

If we encounter the situation with very infrequent demands, the strategy above simply would not work, because the backward communication of the value of having inventory is too slow. We can overcome this by doing a full forward pass, followed by a backward traversal, not unlike what we saw in our shortest path problem in Eq. (48). The only difference is that we would have to compute marginal values during this backward pass. However, it highlights how easily the same problem can do well with one algorithm for one set of parameters (high demands) and very poorly with another set of parameters (low demands).

Approximating value functions with basis functions

Yet another strategy that we could use is to replace the value function with a linear regression model, as we suggested in Eq. (22). If we use approximate value iteration, we would compute a sample estimate of the value of being in state S_t^n using

$$\hat{v}_t^n = \min_{x \in \mathcal{X}} \left(C(S_t^n, x) + \gamma \sum_{f \in \mathcal{F}} \theta_{tf}^{n-1} \phi_f(S_t^x) \right), \tag{59}$$

where as before, S_t^x is the post-decision state variable that depends deterministically on S_t^n and the action x. We might use as basis functions:

$$\begin{aligned}
\phi_1(S_t^x) &= 1, \\
\phi_2(S_t^x) &= R_t^x, \\
\phi_3(S_t^x) &= (R_t^x)^2, \\
\phi_4(S_t^x) &= p_t, \\
\phi_5(S_t^x) &= p_t^2, \\
\phi_6(S_t^x) &= p_t R_t^x.
\end{aligned}$$

As always occurs with parametric models, designing the basis functions is an art form that has to be exercised with care.

We can use the observation $\hat{v}_t^n$ to update our estimate of the regression parameters θ^{n-1} using recursive least squares (Hastie et al. 2009; Powell 2011 [Chap. 7]). This strategy has attracted considerable attention as a potential method for solving the curses of dimensionality (Bertsekas and Tsitsiklis 1996; Tsitsiklis and Roy 1996), but convergence proofs require highly restrictive assumptions, and the algorithmic strategy has been known to diverge, or to simply produce very poor solutions. The difficulty is that it is very hard to evaluate the solution quality without a benchmark.

A deterministic lookahead policy

Now consider a slight variation on our inventory problem. Imagine that we are managing the inventory of trailers at a truck terminal for a parcel service company.

We are going to complicate our problem by introducing the very real dimension that major shippers such as a large big-box retailer may run sales, requiring dozens of trucks on a particular day. These sales are planned, and are communicated to trucking companies so they can plan accordingly. Incorporating this information produces a forecast $f_{tt'}^D$ made on day t of the demand for trailers on day t'.

A forecast represents information that has to go into the state variable, including the post-decision state variable. For example, if we are given a vector $f_t^D = (f_{tt'}^D)_{t' \geq t}$, we might expect it to evolve over time according to

$$f_{t+1,t'}^D = f_{tt'}^D + \hat{f}_{t+1,t'}^D,$$

where $\hat{f}_{t+1,t'}^D$ would capture updates to forecasts (e.g. the retailer may revise their estimates of the number of trailers upward or downward as they get close to the sale). For this reason, the post-decision state now becomes

$$S_t^x = (R_t^x, p_t, f_t^D).$$

Suddenly, our inventory problem has made the transition from a state variable with a relatively small number of discrete states (the different values of R_t^x and p_t), to one where we have added the vector of forecasts f_t^D with an exponential explosion in the number of states (the forecasts f_t^D along with the price p_t belong to the information state I_t). As of this writing, we are not aware of an effective way to solve this problem using policy function approximations (such as (q, Q) policies) or value function approximations. The problem is that the value function $V_t^x(S_t^x)$ in this setting is surprisingly complex; it is just not that easy to create an accurate approximation that would reflect the impact of the vector of demand forecasts on the solution.

Practitioners would normally solve problems in this class using a lookahead policy. The most common strategy is to use a deterministic model of the future, where we would solve

$$X^{\text{DLA}}(S_t) = \arg\min_{x_t} \left(c_{tt} x_{tt} + \sum_{t'=t+1}^{t+H} c_{tt'} x_{tt'} \right), \tag{60}$$

subject to constraints for $t' = t, \ldots, t + H$:

$$A_{tt'} x_{tt'} - B_{t,t'-1} x_{t,t'-1} = R_{tt'}, \tag{61}$$

$$D_{tt'} x_{tt'} \leq f_{tt'}^D, \tag{62}$$

$$x_{tt'} \leq u_{tt'}, \tag{63}$$

$$x_{tt'} \geq 0. \tag{64}$$

Here, Eq. (61) might capture flow conservation, while (62) models the effect of customer demands on the system (we cannot receive money for moving a load twice). Let $x_t^* = (x_{tt'}^*)_{t' \geq t}$ be the optimal solution to (60). Normally, we only retain x_{tt}^* to execute now, while $x_{t,t+1}^*, \ldots, x_{t,t+H}^*$ represents a plan of future decisions that will be updated as new information arrives.

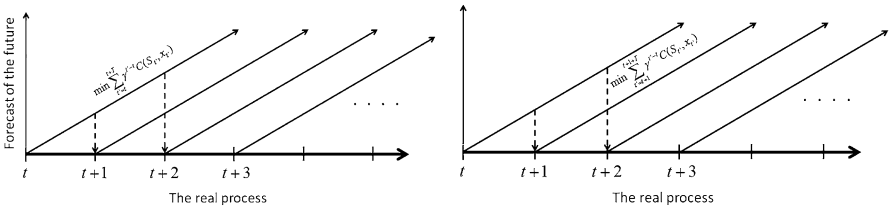

Fig. 2 Illustration of rolling horizon procedure, using a deterministic model of the future

Deterministic lookahead policies are widely used in engineering practice. These are known as rolling horizon procedures, receding horizon procedures and, in engineering, model predictive control. The "problem" with this approach is that they ignore the possibility that the future is likely to be different than our deterministic forecast, and we would like to consider this when making a decision now. The hard part is understanding when a "stochastic model" adds value.

We have to first emphasize that even if we are using a deterministic lookahead policy, we are still using a stochastic model. After all, we can simulate the policy $X^{\text{DLA}}(S_t)$ in a stochastic simulator, as depicted in Fig. 2. It is very important to recognize that there are two models in this process: the real model, which we represent using $S_{t+1} = S^M(S_t, x_t, W_{t+1})$ where $x_t = X^{\text{DLA}}(S_t)$ and where outcomes of W_t are drawn from Ω, and an approximate model that we use purely for the purpose of planning. For this purpose, it is useful to create a new process that we describe using the state variable $\tilde{S}_t$, decision variable $\tilde{x}_t$, information process $\tilde{W}_t$ and transition function $\tilde{S}_{t+1} = \tilde{S}^M(\tilde{S}_t, \tilde{x}_t, \tilde{W}_{t+1})$. For obvious reasons, we refer to this as the "tilde process" which describes the *within-policy model*.

We can introduce a range of approximations in our within-policy model. We might use an aggregated state variable, and we would normally let $\tilde{\Omega}$ be a small set of samples from Ω. When we use a deterministic model of the future, we are approximating the random variables $(W_t, W_{t+1}, \ldots, W_{t+H})$ with a series of point forecasts $(\bar{W}_{tt}, \bar{W}_{t,t+1}, \ldots, \bar{W}_{t,t+H})$ which include, for example, our demand forecasts f_t^D. We no longer have the dynamics of the evolution of demands and prices, and our transition function is now reduced to a system of linear equations such as those represented by Eq. (61). Below, we illustrate policies that introduce other types of approximations.

A modified deterministic lookahead policy

While deterministic lookahead policies are simple and practical, often there is a desire to make decisions now that capture the fact that the future will not match the forecast. Some refer to these policies as *robust*, although this is a term that is interpreted in very specific ways by different communities. Another term is *anticipatory*, recognizing that this could refer to any nonmyopic policy.

A simple way to capture uncertainty is to simply introduce tunable parameters that make the adjustments that would be needed to handle uncertainty. For inventory problems, a common strategy to handle uncertainty in demands is to introduce

buffer stocks, or to replace the expected demand forecast $f_{tt'}^D$ with a quantile (we may decide it is better to aim at the 70th percentile). So, we might replace Eq. (62) with

$$D_{tt'} x_{tt'} \leq f_{tt'}^D + \theta.$$

We might use a single scalar buffer θ, or we could make it time dependent by using $\theta_{t'}$, or a function of how far we are forecasting into the future, as in $\theta_{t'-t}$. An alternative is to replace the point forecast $f_{tt'}^D$ with a quantile $f_{tt'}^D(\theta)$ which is the value that satisfies

$$\mathbb{P}[D_{tt'} \leq f_{tt'}^D(\theta)] = \theta.$$

We can represent either policy using the function $X^{\text{DLA}}(S_t|\theta)$, where θ is now a tunable parameter. We perform the tuning by solving the stochastic optimization problem

$$\min_{\theta} F^{\text{DLA}}(\theta) = \mathbb{E}^{\pi} \sum_{t=0}^{T} C(S_t, X^{\text{DLA}}(S_t|\theta)). \tag{65}$$

This problem is identical to our original objective function in Eq. (15). All that we have done is replace the generic search over policies $\pi \in \Pi$ with a search over θ, which parameterizes a policy. This is simply different notation representing the same thing.

A lookahead policy based on stochastic programming

Our modified rolling horizon policy is a way of adapting to the effect of uncertainty. While this may work quite well for specific problems, we may be left wondering if there is an even better solution. For this reason, a substantial community has evolved under the umbrella of *stochastic programming* which attempts to approximate the future while simultaneously capturing the effect of uncertainty. The easiest way to visualize solving the full stochastic program is to create a decision tree, which explicitly models the sequence of decisions and information. These are hopelessly intractable for the high-dimensional problems that arise in transportation.

An alternative strategy is to create a *scenario tree* of just the random events. For our simple inventory problem, assume we have a single random variable (demand), and that we can discretize the demand into three outcomes: higher than normal, normal, and lower than normal. We can create a tree of all these possible outcomes over the next five days, as depicted in Fig. 3a. It is easy to see that the tree grows quickly, and this idea would never scale if we had more than one random variable (we might have a vector of demand types, along with random prices and costs).

An alternative strategy is to use Monte Carlo sampling, which we have depicted in Fig. 3b. Here, we would use a set of samples $\tilde{\Omega}$ of our information process, where we can control how many samples we want to use to represent the different types of outcomes of the random variables. The important feature here, however, is that we

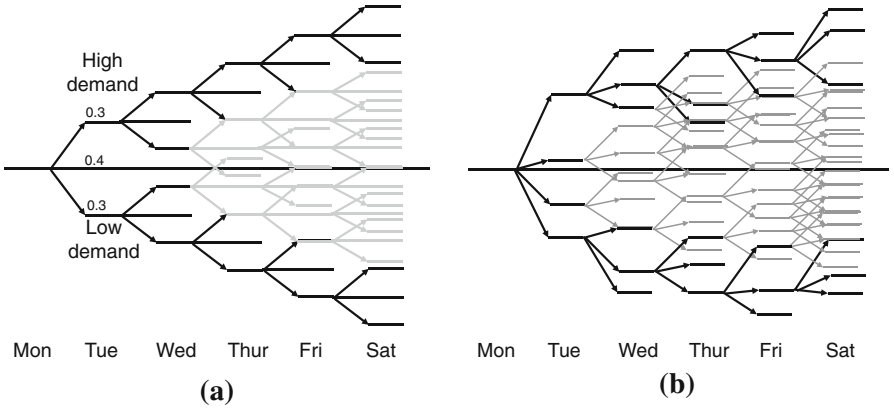

Fig. 3 Illustration of a scenario tree based on a discretized distribution (**a**), and Monte Carlo samples (**b**)

can have any number of random variables; we just have to limit how many samples we use.

Not surprisingly, even this strategy can produce an extremely large scenario tree as the number of time periods (stages) grow. For this reason, it is common practice to use a reasonable number of samples in the first period, and then sharply limit the number of samples in later periods, with the idea that these periods are less important. In fact, many authors take this idea one step further and simply model the entire future as one stage. So, we may represent time periods $(1,\ldots,T)$ as a single stage which is realized all at the same time. This means we have to make a decision x_0 without knowing what might happen in the future. However, $x_1,\ldots,x_T$ are then chosen knowing the entire future $W_1,\ldots,W_T$. This is known as a two-stage approximation, and is written mathematically as

$$X^{\text{SP}}(S_t|\theta) = \arg\min_{x_t}\left(c_{tt}x_{tt} + \sum_{\tilde{\omega}\in\tilde{\Omega}} p(\tilde{\omega}) \sum_{t'=t+1}^{t+H} c_{tt'}(\tilde{\omega})x_{tt'}(\tilde{\omega})\right), \tag{66}$$

subject to constraints for time t:

$$A_{tt}x_{tt} = R_{tt}, \tag{67}$$

$$D_{tt}x_{tt} \le f_{tt}^{D}, \tag{68}$$

$$x_{tt} \le u_{tt}, \tag{69}$$

$$x_{tt} \ge 0, \tag{70}$$

and constraints for $t' = t+1,\ldots,t+H$ and each sample path $\tilde{\omega}\in\tilde{\Omega}$:

$$A_{tt'}(\tilde{\omega})x_{tt'}(\tilde{\omega}) - B_{t,t'-1}(\tilde{\omega})x_{t,t'-1}(\tilde{\omega}) = R_{tt'}(\tilde{\omega}), \tag{71}$$

$$D_{tt'}(\tilde{\omega})x_{tt'}(\tilde{\omega}) \le f_{tt'}^{D}(\tilde{\omega}), \tag{72}$$

$$x_{tt'}(\tilde{\omega}) \le u_{tt'}(\tilde{\omega}), \tag{73}$$

$$x_{tt'}(\tilde{\omega}) \ge 0. \tag{74}$$

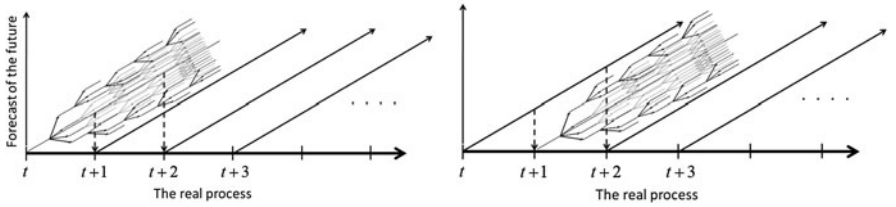

Fig. 4 Illustration of rolling horizon procedure, using a stochastic model of the future

As with our modified deterministic lookahead policy, we have written our stochastic programming policy as a function of a parameter vector θ. In this setting, θ might represent the parameters that determine how the scenario tree is constructed. We can then simulate the policy $X^{\text{SP}}(S_t|\theta)$ just as we simulated $X^{\text{DLA}}(S_t|\theta)$ in Eq. (65) which, in principle, could be used to tune the parameter vector θ. Figure 4 illustrates the process of executing our stochastic programming policy on a rolling horizon basis.

The stochastic programming policy represents the clearest illustration of two models of a process: the "within-policy" model where we represent the future in an approximate way, purely for the purpose of making a decision, and the real process. In many applications, the real process is something that is experienced with the actual passage of time in a physical system. However, our real process can be a computer simulation, but one that uses a much higher level of detail than the within-policy model. At a minimum, in the real process we would experience the full range of random outcomes, in sharp contrast with the highly stylized representation of the future used in our scenario tree (depicted in Fig. 3b).

Fleet management problems

We now make the transition to managing large fleets of vehicles in the context of truckload trucking. In the US (and this seems to be a phenomenon unique to the US) there are companies dedicated to the movement of full truckloads. The three largest megacarriers in the US manage fleets with over 15,000 drivers. The problem is to determine which driver to assign to each load at a point in time, capturing the fact that decisions now have an impact on the future. We start with the problem of managing a single truck.

Optimizing a single truck

Assume that we are managing a single truck at location $i \in \mathcal{I}$. When the truck arrives at i, he is offered a set of loads represented by $\hat{D}_{tij}$ where $\hat{D}_{tij} = 1$ if there is a load that the driver can move from i to j starting at time t. The truck may choose to reject all the loads and remain where he is until time $t + 1$. Any loads that have not been moved at time t are assumed taken by other drivers.

This problem is very similar to our stochastic shortest path problem. We can write the state variable as $S_t = (L_t, \hat{D}_t)$, where L_t is the current location of the truck and $\hat{D}_t$ is the vector indicating the loads available to be moved from his current location to other cities. The vector of loads $\hat{D}_t$ plays a role that is very similar to the random costs in our stochastic shortest path problem. This problem can be easily solved using Bellman's equation if we exploit the post-decision state variable.

Now let's see what happens when we make a subtle change to the problem. Instead of managing a truck characterized only by its location within a set $\mathcal{I}$, assume that there is a driver in the truck. This complicates our resource incredibly, since the driver is now characterized by a vector $r_t = (r_{t1}, r_{t2}, \ldots, r_{tK})$, where the elements r_{tk} include location, equipment status, fuel level, estimated time of arrival, driver type, driver domicile, the number of days the driver has been on the road, and an eight-dimensional vector capturing how many hours the driver worked each of the last eight days (see Simao et al. 2009 for a complete description).

With this new formulation, instead of a driver being at location L_t (of which there may be hundreds or thousands, depending on how we discretize space), he is instead in a state r_t, where the number of possible values that r_t can take on is on the order of 10^{20}. When we solved our shortest path problem, we made the implicit assumption that we could loop over all the nodes in the network performing elementary calculations. This is not possible if the number of nodes is equal to 10^{20}.

We have to circumvent this problem by replacing our lookup table which estimates the value of being at a node in our network with a statistical approximation of the value of a driver having attribute r_t. A powerful way of doing this is to estimate the value of a driver at different levels of aggregation. Let $\bar{V}^g(r)$ be an estimate of the value of a driver with attribute r using aggregation level g. For example, we might have an estimate of the value of a driver at location i as one (very coarse) level of aggregation. More disaggregate estimates might include a joint estimate of location and driver domicile. We can then write our estimate of the value of a driver with attribute r as

$$\bar{V}(r) = \sum_{g \in \mathcal{G}} w^g(r) \bar{V}^g(r), \tag{75}$$

where $\mathcal{G}$ is our set of different aggregation levels, and $w^g(r)$ is the weight we put on the gth level of aggregation for a driver with attribute r. For more details, see (George et al. 2008; Powell 2011 [Chapter 7]). As with the shortest path problem, we encounter the issue of exploration vs. exploitation, although this is mitigated somewhat if we use the hierarchical aggregation strategy in Eq. (75), since we can observe the value of a driver with attribute r and use this to simultaneously improve our estimate of drivers with other attributes. Mes et al. (2011) presents an algorithmic strategy called the knowledge gradient (Powell and Ryzhov 2012) as a potential strategy for solving the exploration-exploitation problem more explicitly. However, as of this writing, the identification of policies to solve the exploration-exploitation problem in this setting remains a very active area of research.

Load matching using a myopic policy

We now make the transition to large fleets of vehicles. Let R_{tr} be the number of drivers with attribute vector r (when r is complex, the number of drivers is usually 0 and sometimes 1). Also let $D_{tb} = 1$ if there is a load with attribute vector b, which includes attributes such as origin, destination, pickup window, delivery window and other attributes such as whether the load requires special equipment. Also let c_{rb} be the cost of assigning a driver with attribute r to a load with attribute b. In this formulation, the revenue from covering the load is used as a negative cost.

This is a rare example of a problem where a myopic policy works reasonably well. This requires solving an assignment problem of the form

$$X^M(S_t) = \arg\min_{x_t} \sum_{r\in\mathcal{R}} \sum_{b\in\mathcal{B}} c_{rb} x_{trb}, \tag{76}$$

subject to

$$\sum_{b\in\mathcal{B}} x_{trb} \le R_{tr}, \tag{77}$$

$$\sum_{r\in\mathcal{R}} x_{trb} \le D_{tb}, \tag{78}$$

$$x_{trb} \ge 0. \tag{79}$$

While myopic policies work well in practice, this decision function suffers from several limitations. Below, we use a myopic cost function approximation to overcome a problem that arises when we do not cover a load now, and delay it to some point in the future. Then, we use a value function approximation to allow our assignments to look into the future.

Load matching using a myopic cost function approximation

Imagine a load that has been delayed several hours, just because there are other loads that are closer and therefore cost less to cover. The customer may tolerate a delay of a few hours, but after this we are starting to face a serious service failure.

An easy way to overcome this problem is to introduce a bonus to cover a load. Let τ_{tb} be the number of time periods that our load with attribute b has been delayed (the delay would actually be one of the attributes). We might solve this problem using the modified policy

$$X^{\text{CFA}}(S_t|\theta) = \arg\min \sum_{r\in\mathcal{R}} \sum_{b\in\mathcal{B}} (c_{rb} - \theta\tau_{tb}) x_{trb}, \tag{80}$$

$$= \arg\min \sum_{r\in\mathcal{R}} \sum_{b\in\mathcal{B}} C^{\pi}(S_t, x_t|\theta). \tag{81}$$

We refer to $C^{\pi}(S_t, x_t|\theta) = \sum_{r\in\mathcal{R}} \sum_{b\in\mathcal{B}} (c_{rb} - \theta\tau_{tb}) x_{trb}$ as a *myopic cost function approximation*. As before, θ is a tunable parameter which can be optimized through repeated simulations of the policy $X^{\text{CFA}}(S_t|\theta)$.

Load matching using value function approximations

This same problem can also be solved using value function approximations to address the limitation that we are ignoring the impact of assigning a particular type of driver to a load. For example, the closest driver to a load headed to New York City (on the east coast of the US) may find that there are no loads available out of New York that would allow the driver to get him back to his home in Chicago (in the middle of the country).

We can solve this problem using the same approximation strategy that we used to solve our single driver problem in "Optimizing a single truck". Assume that we first compute $r' = R^M(r, b)$ which is the attribute vector produced if we assign a driver with attribute vector r to a load with attribute vector b. Using our approximation of the value of a driver in the future, we would then solve

$$X_t^{\text{VFA}}(S_t) = \arg\min \sum_{r\in\mathcal{R}} \sum_{b\in\mathcal{B}} \big(c_{rb} + \bar{V}_t(R^M(r, b))\big)x_{trb}, \tag{82}$$

subject to constraints (77)–(79). For this problem, rather than compute $\hat{v}_t^n$ as the contribution of assigning a driver to a load plus the downstream value [as we did in our stochastic shortest path problem in Eq. (46)], we use the dual variable for the driver assignment constraint given by Eq. (77). We note that in the domain of resource allocation problems, it is quite common to learn slopes of the cost function rather than its value.

This algorithmic strategy has been implemented very successfully for a major truckload carrier, as described in Simao et al. (2009) and Simao et al. (2010). This work uses the same approximate value iteration strategy described in Fig. 1, which allows us to execute the policy as a simple simulator, learning as we go. There is one issue that deserves some discussion. We made the point in "Optimizing a single truck" when we developed a model to optimize a single truck driver that we had to address the exploration-exploitation problem, which is an active area of research. While it is possible to design explicit exploration policies for a single driver, these strategies are not scalable to vector-valued actions. How were we able to circumvent this issue when we made the transition to working with large fleets? The answer is simple; when modeling thousands of drivers, the dynamics of the system forces a certain amount of natural exploration.

Managing fleets of vehicles

We can extend our problem one more time for applications where we are managing fleets of trailers or containers. This is just like managing a fleet of truck drivers, but now the attribute vector r is much simpler, typically consisting of the location of the equipment and the equipment type. The difference between managing fleets of drivers and fleets of vehicles is that drivers are complex, which means that the value of the resource variable R_{tr} is typically 0 or 1. When the attribute vector is simple, such as the number of refrigerated trailers at a location, R_{tr} can be dozens to hundreds. This problem is more like a spatial inventory problem, whereas load matching is more like a dynamic routing and scheduling problem.

The problem of managing fleets of vehicles (rather than drivers) has received considerable attention from the literature. Reviews of this problem class can be found in Crainic and Gendreau (1993), Crainic and Laporte (1997) and Powell et al. (1995). The most common models used in industry are myopic (see Gorman et al. 2011 for a summary of car distribution models at two major railroads) or they use a deterministic lookahead (rolling horizon) policy (Joborn et al. 2004). A number of authors have experimented with various approximations of stochastic models (Jordan and Turnquist 1983; List et al. 2003; Powell and Topaloglu 2006; Lam et al. 2007). Powell and Topaloglu (2005) describes an actual implementation of a model at Norfolk Southern for empty car distribution using a policy based on value function approximations; Powell et al. (2012a, b) describes a model for locomotive management that was also implemented at Norfolk Southern that also uses value function approximations. However, the operational implementation of models that explicitly account for uncertainty are rare.

When we managed individual drivers, we only needed to estimate the marginal value of a single driver in the future. When we work with a more compact attribute space, we need to capture the nonlinearity of the problem with respect to the number of vehicles in a location. We can do this using basically the same strategy that we introduced in "Exploiting convexity" for our inventory problem. Let $R_t^x = (R_{tr}^x)_{r\in\mathcal{R}}$ where R_{tr}^x is the number of vehicles with attribute r produced by making decision x (this is our post-decision resource state vector, which ignores random arrivals, failures and delays). To simplify our presentation a bit, we are going to assume that our only attribute is a location which we index by i, j or k.

As we did in our inventory problem in Eq. (55), we are going to approximate the value of inventories in the future using a piecewise linear, separable, convex function. The only difference is that now we are going to index our slopes $\bar{v}_{tk}^{n-1}(r_k)$ by the location k. For this model, we divide the decision vector $x_t = (x_t^L, x_t^E)$ into loaded and empty movements. To retain our minimization model, we assume that vehicles move empty from location i to j at a cost c_{ij} while trucks moving loaded incur a "cost" of $-r_{ij}$. Our policy can now be written

$$X^{\mathrm{VFA}}(S_t) = \arg\min_{x\in\mathcal{X}}\left(\sum_{i,j\in\mathcal{I}}\left(c_{ij}x_{tij}^E - r_{ij}x_{tij}^D\right) + \gamma\sum_{k\in\mathcal{I}}\sum_{r_k=0}^{R^{\max}}\bar{v}_{tk}^{n-1}(r_k)y_{rk}\right), \tag{83}$$

subject to, for all $i, j, k \in \mathcal{I}$:

$$\sum_{j\in\mathcal{I}}(x_{tij}^L + x_{tij}^E) = R_{ti}^x, \tag{84}$$

$$\sum_{i\in\mathcal{I}}(x_{tij}^L - x_{tij}^E) - R_{tk}^x = 0, \tag{85}$$

$$\sum_{r=0}^{R^{\max}} y_{rk} - R_{tk}^x = 0, \tag{86}$$

$$R_{t+1,i} = R^x_{ti} + \hat{R}_{t+1,i}, \tag{87}$$

$$x^L_{tij} \leq D_{tij}, \tag{88}$$

$$x^L_{tij}, x^E_{tij}, y_{tk} \geq 0. \tag{89}$$

In this problem, the state $S_t = (R_t, D_t)$ captures the distribution of vehicles in R_t, and the available loads to be moved in D_t. Assuming unsatisfied demands are not carried to the next time period, the post-decision state is given by $S^x_t = R^x_t$.

The objective function in Eq. (83) is just a multidimensional extension of our original inventory problem. It helps that the optimization problem (83)–(89) is a linear program, which makes it fairly easy to solve even for industrial scale problems. We note that while maintaining convexity was useful for our inventory problem, it is critical here since we would like to solve the optimization problem using a linear programming package. With our inventory problem, we estimated our slopes $\hat{v}^n_t$ using numerical derivatives [as we did in Eq. (56)]. Here, we would estimate the slopes $\hat{v}^n_{tk}$ using the dual variable for the constraint (84). Note that it is very important that we compute a dual variable for each location i. We can do this easily when the only attribute of a vehicle is its location; we were not able to do this when matching complex drivers in "Load matching using value function approximations"

Our formulation above can be generalized significantly by replacing location $i \in \mathcal{I}$ with attribute $r \in \mathcal{R}$, assuming that the size of $\mathcal{R}$ is not too large (if it is large, then we return to the problem addressed in "Load matching using value function approximations"). We need to be able to compute the dual variable $\hat{v}^n_{tr}$ for each $r \in \mathcal{R}$. This is manageable if $|\mathcal{R}|$ is in the thousands, and possibly tens of thousands, but no larger. With this notation, we can handle multiple equipment types and multi-period travel times, although some care has to be used when generalizing the problem in this way. As with our inventory problem, we can use a simulation-based strategy similar to the algorithm described in Fig. 1, which is very easy to implement and scales to industrial-strength problems.

Dynamic vehicle routing

All of the problems above can be viewed as just a warmup for dynamic vehicle routing, arguably one of the hardest problems in operations research. Dynamic vehicle routing has received considerable attention, and since we are using the setting only to illustrate the different types of policies, we refer to Larsen (2000) and Pillac et al. (2011) for thorough reviews. Given the complexity of solving static vehicle routing problems (Toth and Vigo 2002), there has been considerable attention devoted to solving sequences of deterministic problems (routing vehicles among customer requests that are already known) quickly enough, either using partial re-optimizations or parallel computation (as in Gendreau et al. 1999). While such deterministic lookahead policies allow for the direct use of existing algorithms, the research community has recognized that there is a need for algorithms that

recognize customer orders that might become known, leading to behaviors where vehicles are kept close to groups of customers where orders are likely to arise.

A number of papers have explored policies which account for an uncertain future. Potvin et al. (2006) and Ichoua et al. (2006) investigate classes of policy function approximations, which introduces rules for holding trucks to allow for orders that have not yet become known (rules that determine an action are a form of lookup table policy function approximation). Similar algorithmic strategies are referred to as *online algorithms* which often (though not always) are implemented as simple rules that can be easily computed in real time to react to new information as it arrives (see Van Hentenryck et al. 2009 and Van Hentenryck and Bent 2009 for thorough discussions of online stochastic optimization).

For complex transportation applications (such as dynamic vehicle routing), it is generally necessary to plan activities into the future. The most common approach is to optimize vehicle tours over all known customers. This is a class of rolling horizon procedure, but one that does not even use forecasted customers (because customer orders are 0/1, it does not make sense to plan a vehicle tour to a potential customer that might happen with probability .2). This is a fairly serious limitation, because there may be groups of customers with no known orders at a point in time, but where it is likely that orders within the group will arise during the day. Dispatchers know this and plan accordingly, keeping trucks in a region where orders are likely to arise, without knowing which customer will generate the order.

A strategy that overcomes this limitation is to generate scenarios of potential outcomes in the future, just as we depicted in Eqs. (66)–(70). This is the strategy used in Bent and Van Hentenryck (2004) and Hvattum et al. (2006) where multiple scenarios of future customer orders are used to improve decisions now. Recall that the policy in Eq. (66), denoted by $X^{\mathrm{SP}}(S_t|\theta)$, can be tuned with different rules for how these scenarios are generated. While it is tempting to generate as many scenarios as possible, increasing the number of scenarios produces diminishing returns. Assume, for example, that θ controls the number of scenarios that are generated (it could also control how the scenarios are generated). We can optimize over the scenario generation policy θ by solving

$$\min_{\theta} \mathbb{E} \sum_{t=0}^{T} C\big(S_t, X^{\mathrm{SP}}(S_t|\theta)\big). \tag{90}$$

Of course, this has to be done with Monte Carlo sampling, but the idea is that we evaluate our scenario generating policy based on how well it works in simulations. This idea is recognized in Bent and Van Hentenryck (2004) and Mercier and Van Hentenryck (2011), with additional experiments and testing in Schilde et al. (2011) which support the effectiveness of these ideas. Of course, a stochastic lookahead policy such as $X^{\mathrm{SP}}(S_t|\theta)$ is going to be computationally demanding [there is considerable research accelerating the calculation of *deterministic* lookahead policies, as in Gendreau et al. (1999)], motivating the continued search for robust, computable policies. We use this challenging problem purely as a way of illustrating the potential of a hybrid policy.

For our discussion, we let $\hat{D}_{tt'i}$ be the demand at location i, to be served at time t' or later (but as close to time t' as possible) that first became known at time t. We let $D_{tt'i}$ be the demands known at time t to be served at time t' or later at location i, that may have become known at time t or earlier. We assume that our vehicle(s) may be re-optimized at each time t as new information becomes available. The vector $D_{0t'}$ is the set of customer orders that are known at the beginning of the day.

If this is a pickup problem, the vehicle may need to return to the depot to drop off product if the vehicle hits capacity; if it is a delivery problem, the vehicle may have to return to the depot to pick up new product. We may plan a trip into the future, but we only communicate the next step to the driver, and we may replan trips as new information becomes available. We assume that we are dealing with a single product (this is easiest to envision with pure pickup problems, where we only care about how much space is used). This means that the state of a truck is its location and how much product is on the truck at time t. If $\mathcal{I}$ is our set of locations, let

$$L_{ti} = \begin{cases} 1 & \text{If there is a truck at location } i \text{ at time } t, \\ 0 & \text{Otherwise,} \end{cases}$$

$$F_{ti} = \text{ The total amount of product on a truck at } i \text{ at time } t \ (=0 \text{ if } L_{ti} = 0).$$

The state of our resources (trucks) can now be written $R_t = (L_t, F_t)$ where L_t and F_t are vectors over all the locations $i \in \mathcal{I}$. This allows us to write the state variable as

$$S_t = (R_t, D_t),$$

where $D_t = (D_{tt'i})_{t' \geq t, i \in \mathcal{I}}$ are the unserved demands known at time t. This representation captures our need to model both the resources and the demands. In the past, we simplified our state variable by exploiting the fact that the post-decision state did not include information about demands, which allowed us to design value function approximations around the resource vector R_t. This sleight of hand is not possible with dynamic vehicle routing.

This problem shares elements of all the problems we have discussed above, but introduces new complications. The two aspects that seem to create the greatest difficulty is the presence of advance information about demands in the form of the vector $D_{tt'}$ and, most problematically, that we now have to represent both resources and demands in the state variable. In our applications above, we were able to handle vector-valued resource variables R_t by creating value function approximations that are linear [as in Eq. (82)], or piecewise linear separable [as in Eq. (83)] in the resource variable.

A reasonable question might be: if we introduce demands in the state variable, why can't we consider modeling them just as additional dimensions? For example, we could have a linear coefficient for the value of a demand, or perhaps have a nonlinear function for unserved demands. While it would be interesting to see this strategy tested, there are some reasons why this approach is unlikely to succeed. First, the marginal effect of an additional demand to be served in the future depends on the specific routing of vehicles, which is highly dependent on the entire vector of demands. This obvious fact makes it unlikely that any sort of separable approximation would be effective.

Second is the problem of mixing resources and demands]known as a two-layer resource allocation problems in the language of Powell et al. (2001)]. Separable approximations for the value of additional resources may not be perfect, but it appears to be a surprisingly robust strategy in the settings where it has been tested up to now. Of course the value of an additional driver at node i of our network depends on the location of other drivers, but over time the value function approximation for one dimension learns about the likely values of other dimensions. The same property is unlikely to be true when we mix resources and demands. The value of a resource may be small or zero if there are no demands to be served nearby. The value of a demand may be small or zero if there are no vehicles nearby. But if there is a vehicle and a demand, then we gain value. The non-separability of resources and demands seems to be difficult to ignore.

A potential policy would be to use a hybrid lookahead with value function approximation. The idea is depicted in Fig. 5. Imagine that we are routing two trucks (labeled A and B), and that at time t we have seven known customer orders, depicted by the larger circles. Assume we have managed to calculate an approximation of the value of a truck at each customer location. We route the truck by allowing the tour to terminate at any customer location, at which point it would incur the cost represented by the value function approximation, or it may continue to another downstream customer (in which case we ignore the value

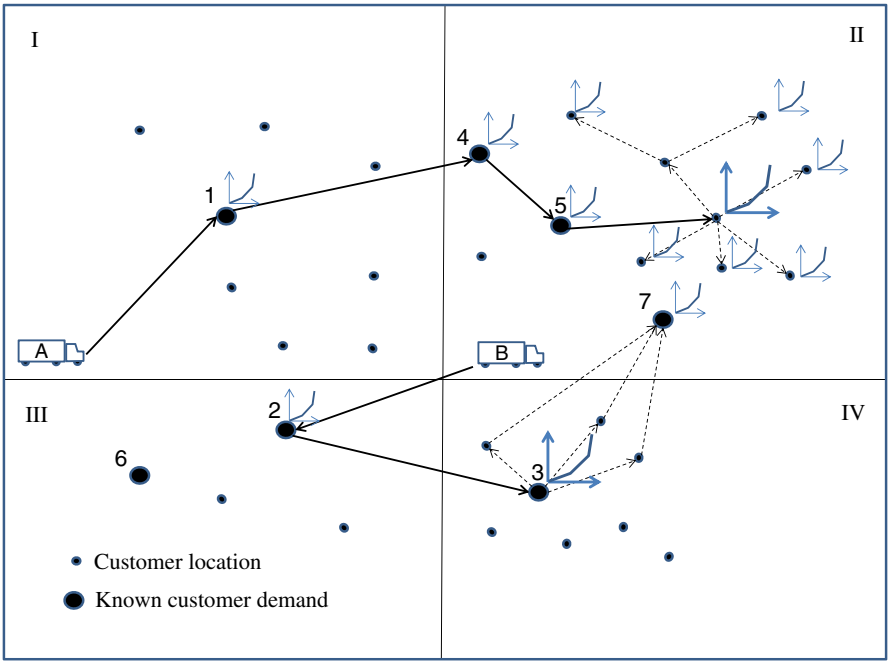

Fig. 5 A possible solution routing two trucks to serve up to seven known customers, among a large set of potential customers. The vehicle may stop at any location and receive the benefit captured by the value function approximation (depicted as a piecewise linear function), or continue to the next customer

function at that location). Such a strategy is easily handled using any standard routing heuristic. We would use this solution only to determine the next customer a truck should visit, since new customer orders may become known by the time it makes its next stop.

A possible solution is given in Fig. 5, which shows truck A serving customers 1, 4 and 5, and then making its way to a customer location with no order, but which is in the midst of a number of other potential customers. The truck could have terminated at customer 5, but we presume that it was less expensive to proceed to the uncovered customer that is in a more central location. The dashed lines show possible trajectories of the truck after stopping in the final location; the expected value of these subsequent assignments are approximated by the value function approximation at the location where the vehicle stopped.

The figure also shows truck B serving customer 2 and then customer 3. The more natural tour (barring time window constraints) would have been to serve customer 3 first, but the tour shown in the figure captures the higher value of terminating at customer 3, which is in the midst of a cluster of other customers, and is therefore of higher value.

Note that our planned truck tours do not cover all the customers, as neither customers 6 or 7 are currently being covered. The figure illustrates that after serving customer 3, the vehicle might first serve other customers in the region (with orders that are not yet known) and then serve customer 7. Even though we know about customer 7 now, this order can still be captured in the value function when the vehicle stops at customer 3. Thus, we need to consider solutions which intermingle known and unknown customers.

A sketch of the mathematical problem works as follows: Let x_t^k represent the tour of vehicle k given what we know at time t, which covers the path over as many customers as we think should be covered given what we know at time t. We let $X^\pi(S_t)$ be our policy that returns these tours, where S_t captures the current location of all the vehicles as well as the full vector of known customer orders. As before, we are not going to implement the entire tour; we only implement the instruction of what the truck should do next.

Let $\bar{V}_t^{n-1}(S_t^x)$ be the (post-decision) value function approximation, computed after $n-1$ simulations, given what we know at time t, where S_t^x is the state of the system given the tour x. As depicted in Fig. 5, we are using a separable value function approximation, with a function capturing the value of a truck if it ends its tour at each location. The policy is allowed to stop at any location and incur the cost in the value function, or proceed to the next customer location. A tour may stop at a location with uncovered customers; in this case, the value function would have to capture the value of covering other known customers.

Our policy is based on value function approximations which we can write as

$$\tilde{V}_t(S_t^n) = \min_{x_t}\big(C(S_t^n, x_t) + \bar{V}_t(S_t^x(S_t^n, x_t))\big). \tag{91}$$

This optimization would have to be solved using an appropriate VRP heuristic, since these problems typically cannot be solved exactly. It is important to realize that x_t is an entire tour, and not just the assignment of the vehicle to a single

customer (as we did in our load matching example). The post-decision state S_t^x reflects the updated state of the system after taking into account the effect of the tours of all the vehicles for as far into the future as we have planned their movement.

We might estimate the value function approximation using derivative information. There are different ways of obtaining this information, but a brute force approach would be to use

$$\hat{v}_t(i,t') = \tilde{V}_t(S_t + e_t(i,t')) - \tilde{V}_t(S_t)$$

where $e_t(i,t')$ is a vector of 0's with a 1 corresponding to having an additional vehicle at location i at time t'. Of course, obtaining these derivatives may be expensive (in practice we would probably do a local optimization around the solution obtained from solving (91)). The more difficult problem is designing the value function itself. We might capture the value of a single vehicle at a location, or the value of one more vehicle in one of the four aggregate quadrants. We would like to capture the effect of uncovered customers, but one approach would be to aggregate the total number of uncovered customers in a quadrant. The key idea here is that we capture the detailed interactions of drivers and customers by deterministically solving a partial routing problem, and then hoping that a simpler value function approximation will suffice to capture events farther in the future. Needless to say, there is a lot of room for experimentation here.

Choosing a policy

The examples above illustrate a range of different operational problems that arise in transportation and logistics. We have used these problems to illustrate the concept of the post-decision state (which allows us to solve deterministic optimization problems without an imbedded expectation), and four classes of policies: myopic cost function approximation (used in the load matching example), lookahead policies (used for planning inventories in a time-dependent setting with forecasts, and again in our dynamic vehicle routing problem), policy function approximations [such as (q,Q) inventory policies], and policies based on value function approximations (which we used in our simple inventory problem, as well as in managing fleets of vehicles). The dynamic vehicle routing problem in "Dynamic vehicle routing" is a nice illustration of a complex operational problem that is likely going to require some form of hybrid.

Each of these application settings exhibited structure that suggested a particular type of policy. If the policy is not obvious, it is always possible to code up a set of competing policies and compare them using the objective function in Eq. (15). In most cases, any algorithm will involve tunable parameters (denoted by θ above) which will have to be optimized. Policy search covers whether we are comparing major classes of policies (e.g. lookahead versus value function approximation) or tuning the parameters of a policy within a particular class.

Coding and testing a policy requires a considerable amount of time, which leaves the question: How to choose? Based on our experience, we offer the following guidelines:

- A myopic cost function approximation is going to work well if a myopic policy works reasonably well. The load matching problem is a good example of a problem where myopic policies work well, so it is not surprising that introducing some tunable parameters to overcome obvious problems (such as delaying loads), will produce a better policy.
- Deterministic lookahead policies are going to work well when we have good information about the future (such as forecasts of future customer requirements in a time-dependent inventory problem). In an inventory problem, we can imbed a cost function approximation in the form of buffer stocks to make the solution more robust. Such strategies become less obvious in applications such as our dynamic vehicle routing problem, where we cannot modify a deterministic solution using a simple idea such as buffer stocks. One problem with deterministic lookahead policies is that the computational complexity grows quickly with the length of the planning horizon. In a locomotive application, we found that the CPU times grew from a few seconds for a horizon of four hours, to over 50 hours for a horizon of four days.
- Stochastic lookahead policies have attracted considerable attention in operations research under the umbrella of stochastic programming. The attraction is that the optimization problem given by Eqs. (66)–(74) can be solved, in principle, as a single (albeit very large) linear or integer program. It is important to realize that (66)–(74) is only a two-stage approximation of a multistage problem, and yet, for most applications, even this problem is extremely difficult to solve. Imagine using this for a dynamic vehicle routing problem, where we would be optimizing vehicle tours across all scenarios. While this idea has attracted attention in the research community (see, for example, Gendreau et al. 1995; Laporte et al. 2002), we are not aware of any production implementations. We suspect that stochastic programming using scenario trees is best suited to problems with very coarse-grained noise. For example, imagine the problem of analyzing the acquisition of container ships over a 20 year period in the presence of uncertainty about global economies and the price of oil. Scenario trees are a good method for representing this type of uncertainty.
- Policies based on value function approximations - Often when people refer to the use of "approximate dynamic programming" they are referring to policies based on value function approximations. There is actually a fairly simple way to assess the relative strengths of lookahead policies over value functions. First, we write the objective functions for a deterministic lookahead policy,

$$X_t^{\text{DLA}}(S_t) = \arg\min_{x_t} c_{tt}x_{tt} + \underbrace{\sum_{t'=t+1}^{t+H} c_{tt'}x_{tt'}}_{V_t^D(S_t^x)}, \tag{92}$$

and a stochastic lookahead policy,

$$X_t^{\mathrm{SP}}(S_t|\theta) = \arg\min_{x_t} c_{tt}x_{tt} + \underbrace{\sum_{\omega\in\Omega} p(\omega) \sum_{t'=t+1}^{t+H} c_{tt'}(\omega)x_{tt'}(\omega)}_{V_t^S(S_t^x)}. \tag{93}$$

In (92), we have labeled the portion of the objective function that applies to the future as $V_t^D(S_t^x)$ where S_t^x is the state that results from the decision x_{tt}. In (93), we have similarly labeled the stochastic model of the future as $V_t^S(S_t^x)$, where S_t^x plays the same role. Now contrast the policy that uses a value function approximation

$$X_t^{\mathrm{VFA}}(S_t) = \arg\min_{x_t}\big(c_{tt}x_{tt} + \bar{V}_t(S_t^x)\big). \tag{94}$$

As a general rule, Eq. (94) is *much* easier to solve than Eqs. (92) and (93), but the question is whether we can obtain high quality solutions. We can address this question fairly simply. Policies based on value function approximations [Eq. (94)] will work well if the functions $V_t^D(S_t^x)$ or $V_t^S(S_t^x)$ are not too complex. This was the case with our load matching and fleet management problems, where linear [Eq. (82)] and piecewise linear separable [Eqs. (55), (83)] were found to work quite well. But the situation can change quickly, as we found when we introduced demand forecasts into our inventory problem. Suddenly, the state variable jumped from the amount of product in inventory (or the number of trucks at a location), to one which included the vector of forecasts. These forecasts play a role similar to the known customer orders in our vehicle routing problem. They are easy to handle in a lookahead policy, but make the problem much harder to approximate.

Concluding remarks

The goal of our presentation was to offer a range of different stochastic optimization problems arising in transportation and logistics in a coherent, integrated framework. The transportation community has long mastered the art and science of modeling deterministic operational problems using the language of math programming. However, there is a range of styles for modeling stochastic optimization problems, which complicates the process of understanding how different types of policies can be used on the same problem. For this reason, we feel that the most important contribution of this paper is the use of a common modeling framework, with a formal treatment of subtle concepts such as state variables and policies.

From this foundation, we used carefully chosen applications to illustrate problems where different types of policies worked well. At the same time, these problems allowed us to indicate settings where a class of policy may not work well. We closed with a dynamic vehicle routing problem which has attracted considerable attention, but with surprisingly little progress in terms of robust, high quality policies for stochastic, dynamic applications. There is a wide range of issues that arises in the complex problems in transportation and logistics. We believe that

solutions to these problems will build on the four fundamental classes of policies described here.

Acknowledgments This research was supported in part by grant AFOSR contract FA9550-08-1-0195 and the National Science Foundation grant CMMI-0856153.

References

Barto AG, Bradtke SJ, Singh SP (1995) Learning to act using real-time dynamic programming. Artificial Intelligence 72(1–2):81–138

Bellman RE (1957) Dynamic programming. Princeton University Press, Princeton

Bent RW, Van Hentenryck P (2004) Scenario-based planning for partially dynamic vehicle routing with stochastic customers. Oper Res 52:977–987

Bertsekas DP, Castanon DA (1999) Rollout algorithms for stochastic scheduling problems. J Heuristics 5:89–108

Bertsekas DP, Tsitsiklis JN (1996) Neuro-dynamic programming. Athena Scientific, Belmont

Birge JR, Louveaux F (1997) Introduction to stochastic programming. Springer, New York

Braysy O, Gendreau M (2005) Vehicle routing problem with time windows, Part I: route construction and local search algorithms. Transp Sci 39(1):104–118

Crainic T, Gendreau M (1993) Dynamic and stochastic models for the allocation of empty containers. Oper Res 41(1):102–126

Crainic TG, Laporte G (1997) Planning models for freight transportation. Eur J Oper Res 97:409–438

Dantzig GB (1951) Application of the simplex method to a transportation problem. In: Koopmans T (ed) Activity analysis of production and allocation. Wiley, New York, pp 359–373

Dantzig GB, Ferguson A (1956) The allocation of aircrafts to routes: an example of linear programming under uncertain demand. Manag Sci 3:45–73

Ferguson AR, Dantzig GB (1955) The problem of routing aircraft: a mathematical solution. Aeronaut Eng Rev 14:51–55

Gendreau M, Guertin F, Potvin JY, Taillard E (1999) Parallel tabu search for real-time vehicle routing and dispatching. Transp Sci 33:381–190

Gendreau M, Laporte G, Seguin R (1995) An exact algorithm for the vehicle routing problem with stochastic demands and customers. Transp Sci 29:143–155

George A, Powell WB, Kulkarni S (2008) Value function approximation using multiple aggregation for multiattribute resource management. J Mach Learn Res 9:2079–2111

Godfrey G, Powell WB (2002) An adaptive dynamic programming algorithm for dynamic fleet management. II: Multiperiod travel times. Transp Sci 36(1): 40–54

Gorman M, Crook K, Sellers D (2011) North American freight rail industry real-time optimized equipment distribution systems: state of the practice. Transp Res Part C 19(1):103–114

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

Haykin S (1999) Neural networks: a comprehensive foundation. Prentice Hall, USA

Higle J, Sen S (1991) Stochastic decomposition: an algorithm for two-stage linear programs with recourse. Math Oper Res 16(3):650–669

Higle J, Sen S (1996) Stochastic decomposition: a statistical method for large scale stochastic linear programming. Kluwer Academic Publishers, New York

Hvattum LM, Løkketangen A, Laporte G (2006) Solving a dynamic and stochastic vehicle routing problem with a sample scenario hedging heuristic. Transp Sci 40(4):421–438

Ichoua S, Gendreau M, Potvin J-Y (2006) Exploiting knowledge about future demands for real-time vehicle dispatching. Transp Sci 40:211–225

Joborn M, Crainic TG, Gendreau M, Holmberg K, Lundgren JT (2004) Economies of scale in empty freight car distribution in scheduled railways. Transp Sci 38:121–134

Jordan WC, Turnquist MA (1983) A stochastic dynamic network model for railroad car distribution. Transp Sci 17:123–145

Kall P, Wallace S (1994) Stochastic programming. Wiley, New York

Lam S-w, Lee L-h, Tang L-c (2007) An approximate dynamic programming approach for the empty container allocation problem. Transp Res 15:265–277

Laporte G, Hamme LV, Louveaux FV (2002) An integer L-shaped algorithm for the capacitated vehicle routing problem with stochastic demands. Oper Res 50:415–423
Larsen A (2000) The dynamic vehicle routing problem. PhD thesis, Technical University of Denmark.
Le Bouthillier A, Crainic TG (2005) A cooperative parallel meta-heuristic for the vehicle routing problem with time windows. Comp Oper Res 32:1685–1708
List GF, Wood B, Nozick LK, Turnquist MA, Jones DA, Kjeldgaard EA, Lawton CR (2003) Robust optimization for fleet planning under uncertainty. Transp Res 39:209–227
Mercier L, Van Hentenryck P (2011) Amsaa: a multistep anticipatory algorithm for online stochastic combinatorial optimization. Ann Oper Res 184:233–271
Mes MRK, Powell WB, Frazier PI (2011) Hierarchical knowledge gradient for sequential sampling. J Mach Learn Res 12:2931–2974
Pearl J (1984) Heuristics: intelligent search strategies for computer problem solving. Addison-Wesley, Boston
Pillac V, Gendreau M, Guéret C, Medaglia A et al (2011) A review of dynamic vehicle routing problems. Working paper, CIRRELT, University of Montreal
Potvin J, Xu Y, Benyahia I (2006) Vehicle routing and scheduling with dynamic travel times. Comp Oper Res 33(4):1129–1137
Powell WB (2011) Approximate dynamic programming: solving the curses of dimensionality, 2nd edn. Wiley, Hoboken
Powell WB, Godfrey G (2001) An adaptive, distribution-free approximation for the newsvendor problem with censored demands, with applications to inventory and distribution problems. Manag Sci 47(8):1101–1112
Powell WB, Ryzhov I (2012) Optimal learning. Wiley, Hoboken
Powell WB, Simao HP (2009) Approximate dynamic programming for management of high value spare parts. J Manuf Technol Manag 20(2):147–160
Powell WB, Topaloglu H (2005) Fleet management. In: Wallace S, Ziemba W (eds) Applications of stochastic programming. Math Programming Society: SIAM Series in Optimization, Philadelphia, pp 185–216
Powell WB, Topaloglu H (2006) Dynamic-programming approximations for stochastic time-staged integer multicommodity-flow problems. Informs J Comput 18(1):31
Powell WB, Bouzaiene-Ayari B, Cheng C, Fiorillo R, Das S, Lawrence C (2012a) Strategic, tactical and real-time planning of locomotives at Nortfolk Southern using approximate dynamic programming. In: ASME Joint Rail Conference, ASME, Philadelphia
Powell WB, Bouzaiene-Ayari B, Lawrence C, Cheng C, Das S, Fiorillo R (2012b) JRC2012-74187. In: ASME Joint Rail Conference, Philadelphia, pp 1–10
Powell WB, George A, Lamont A, Stewart J (2011) SMART: a stochastic multiscale model for the analysis of energy resources, technology and policy. Informs J Comput
Powell WB, Jaillet P, Odoni A (1995) Stochastic and dynamic networks and routing. In: Ball MO, Magnanti TL, Monma CL, Nemhauser GL (eds) Network routing: handbooks in operations research and management science, vol 8. North-Holland, Amsterdam, The Netherlands, pp 141–295
Powell WB, Ruszczynski A, Topaloglu H (2004) Learning algorithms for separable approximations of discrete stochastic optimization problems. Math Oper Res 29(4):814–836
Powell WB, Simao HP,Shapiro JA (2001) A representational paradigm for dynamic resource transformation problems. in R. In: Coullard FC, Owens JH (eds) Annals of operations research, J. C. Baltzer AG, The Netherlands, pp 231–279
Puterman ML (2005) Markov decision processes, 2nd edn. Wiley, Hoboken
Rockafellar RT, Wets R (1991) Scenarios and policy aggregation in optimization under uncertainty. Math Oper Res 16(1):119–147
Romisch W, Heitsch H (2009) Scenario tree modeling for multistage stochastic programs. Math Program 118:371–406
Schilde M, Doerner KF, Hartl RF (2011) Metaheuristics for the dynamic stochastic dial-a-ride problem with expected return transports. Comp Oper Res 38(12):1719–1730
Shapiro A, Dentcheva D, Ruszczynski A (2009) Lectures on stochastic programming: modeling and theory. SIAM, Philadelphia
Simao HP, Day J, George A, Gifford T, Powell WB, Nienow J (2009) An approximate dynamic programming algorithm for large-scale fleet management: a case application. Transp Sci 43(2):178–197

Simao HP, George A, Powell WB, Gifford T, Nienow J, Day J (2010) Approximate dynamic programming captures fleet operations for schneider national. Interfaces 40(5):342–352
Spall JC (2003) Introduction to stochastic search and optimization: estimation, smulation and control. Wiley, Hoboken
Taillard E, Badeau P, Gendreau M, Guertin F, Potfin J-Y (1997) A Tabu search heuristic for the vehicle routing problem with soft time windows. Transp Sci 31(2):170–186
Topaloglu H, Powell WB (2006) Dynamic programming approximations for stochastic. Time-staged integer multicommodity flow problems. Informs J Comput 18:31–42
Toth P, Vigo D (eds) (2002) The vehicle routing problem, vol 9, SIAM Monographs on Discrete Mathematics and Applications
Tsitsiklis JN (1994) Asynchronous stochastic approximation and Q-learning. Mach Learn 16:185–202
Tsitsiklis JN, Roy B (1996) Feature-based methods for large scale dynamic programming. Mach Learn 22(1):59–94
Van Hentenryck P, Bent RW (2009) Onlinestochastic combinatorial optimization. MIT Press, Cambridge
Van Hentenryck P, Bent RW, Upfal E (2009) Online stochastic optimization under time constraints. Ann Oper Res 177(1):151–183