REINFORCEMENT LEARNING AND STOCHASTIC OPTIMIZATION

A unified framework for sequential decisions

Warren B. Powell

August 22, 2021

WILEY-INTERSCIENCE

A JOHN WILEY & SONS, INC., PUBLICATION

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

Library of Congress Cataloging-in-Publication Data:

Optimization Under Uncertainty: A unified framework
Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

CHAPTER 18

FORWARD ADP III: CONVEX RESOURCE ALLOCATION PROBLEMS

In chapter 3, we introduced general purpose approximation tools for approximating functions without assuming any special structural properties. In this chapter, we focus on approximating value functions that arise in dynamic resource allocation problems where contribution functions (and, as a byproduct, value functions) tend to be convex (concave if maximizing) in the resource dimension. It is standard practice in the optimization community to refer to these problems as "convex" since minimization is standard, but we will stick to our standard practice of maximizing.

For example, if R is the amount of resource available (water, oil, money, vaccines) and $V(R)$ is the value of having R units of our resource, we often find that $V(R)$ will be concave in R (where R is often a vector). Often, it is piecewise linear, whether R is discrete (e.g. inventories of trucks or units of blood) or continuous (as would arise if we are managing energy or money). Value functions with this structure yield to special approximation strategies, and some of the issues we encountered in the previous two chapters (notably the exploration-exploitation problem) vanish.

There is a genuinely vast range of problems that can be broadly described as dynamic resource allocation. Table 18.1 provides just a hint of the diversity of application settings in this domain. Almost all of these settings involve multidimensional decisions, as we manage different resources (doctors, truck trailers, blood), different types of resources (physician specialties, trailer types, blood types), any of which may be spatially distributed.

We are going to begin with a simple scalar problem where R_t is the amount of resource (energy in a battery, cash on hand, inventory of parts) on hand at time t. We then transition to vector-valued problems. These arise in many settings, but we are going to use the context

Major field	Problem	Resource
Energy	Grid operations	Energy generators
	Grid operations	Natural gas supplies
	Grid operations	Energy from wind
	Battery storage	Storage capacity
	Battery storage	Energy in the battery
	Building management	Building temperature
Health	Public health	COVID tests
	Public health	Vaccines
	Public health	Nurses
	Public health	Blood inventories
	Hospitals	ICU capacity
	Hospitals	Physicians
	Hospitals	Nurses
	Hospitals	Medications
	Hospitals	Blood supplies
Logistics	Inventory management	On-hand inventory
	Inventory management	Material handling
	Manufacturing	Stamping machines
	Manufacturing	Robots
	Supply chain	Suppliers
	Supply chain	Raw materials
Freight transportation	Truck operations	Drivers
	Truck operations	Loads
	Truck operations	Trailers
	Rail operations	Locomotives
	Rail operations	Freight cars
	Ocean	Vessels
	Ocean	Port handling capacity
Finance	Trading	Investments
	Trading	Cash
	Trading	Risk exposure
Laboratory sciences	Equipment	Microscopes
	Equipment	Scanners
	Equipment	Computers
	Materials	Oxygen
	Materials	Metals
	People	Scientists
	People	Technicians

Table 18.1 Sample list of resource allocation problems arising in different problem domains.

of spatially distributed problems as our motivating application, where we define

$$\begin{aligned} R_{ti} &= \text{quantity of resource available at location } i \in \mathcal{I} \text{ at time } t, \\ R_t &= \text{the resource state vector,} \\ &= (R_{ti})_{i \in \mathcal{I}}. \end{aligned}$$

Depending on the underlying problem, the spatially distributed problem may be spread over tens, hundreds, or many thousands of locations, creating potentially a very high dimensional

problem. We will use the spatially distributed setting to motivate vector-valued resource state variables, but vector-valued resource allocation problems arise in a variety of settings:

R_{tk} = Quantity of resource of type $k \in \mathcal{K}$ (type of shirt, color) which can be substituted (at a cost) to satisfy a demand $D_{t\ell}$ for products of type ℓ.

$R_{tt'}$ = Resources that we know about at time t that will be available to be used at time t'.

R_{ta} = Resources (such as people or complex equipment) with attribute vector $a = (a_1, a_2, \ldots, a_M) \in \mathcal{A}$.

The notation R_{ta} is the most general, but opens up the door to a potentially very high dimensional resource vector if the attribute vector a has more than two or three dimensions.

We consider a series of strategies for approximating the value function using increasing sophistication:

Piecewise linear, concave - We start with this for a simple, scalar inventory problem to demonstrate the power of concavity.

Separable, piecewise linear, concave - These functions are especially useful when we are interested in integer solutions. Separable functions are relatively easy to estimate and offer special structural properties when solving the optimality equations.

General nonlinear regression equations - Here, we bring the full range of tools available from the field of statistics.

Cutting planes - This is a technique for approximating multidimensional, piecewise linear functions that has proven to be particularly powerful for multistage linear programs such those that arise in dynamic resource allocation problems.

Linear approximations - There are problems where value functions that are linear in the resources can be quite useful, especially for very high-dimensional problems, where the number of resources, say, with attribute vector a is typically 0 or 1.

Resource allocation with an exogenous state variable - All of the approximations up to now consist purely of a resource vector R_t in the state variable. There are problems where we need to capture other information, that we identify by I_t, giving us a state variable $S_t = (R_t, I_t)$, and where we do not enjoy the structure of concavity (or convexity) in I_t.

An important dimension of this chapter will be our use of derivatives to estimate value functions, rather than just the value of being in a state. When we want to determine how much oil should be sent to a storage facility, what matters most is the marginal value of additional oil. For some problem classes, this is a particularly powerful device that dramatically improves convergence.

This chapter will expect the reader has a background in linear programming. We will assume some understanding with the tools for solving linear programs (although no working knowledge of the algorithms is needed). More important will be an understanding of dual variables, which we use for estimating value functions.

18.1 RESOURCE ALLOCATION PROBLEMS

In chapter 8 we presented a number of problems that could be described as resource allocation problems. In this chapter, we are going to use three to illustrate different

algorithmic strategies: our familiar newsvendor problem, a two-stage resource allocation problem with substitution, and finally a very general, multiperiod resource allocation problem.

18.1.1 The newsvendor problem

Perhaps the most elementary resource allocation problem is known as the newsvendor problem, which we first introduced in section 2.3.1. Here, we first allocate a quantity of a resource ("newspapers") x paying a unit cost c per newspaper, then observe a demand D, where we sell the smaller of x and D at a price p.

Newsvendor problems arise throughout stochastic resource allocation problems. For example, a transportation company (a railroad, an airline, a shipping company) often has to place orders for equipment a year or more in advance. The company hopes that all the equipment will be used, and will be enough to satisfy demand. If the company orders too much, it faces an overage situation. If the company has ordered too few, then it is in an underage situation.

In our notation, we would define

x = the order quantity that can be used to satisfy upcoming demands (not yet revealed),
D = the demand that arises during time interval 1,
c = the unit purchase cost of assets,
p = the price for each unit of demand that is satisfied.

Our contribution function is given by

$$F(x) = \mathbb{E}F(x, D) = \mathbb{E}\left\{p \min[x, D] - cx\right\}. \tag{18.1}$$

We assume (as occurs in the real newsvendor problem) that unused assets have no value (as would happen if we were actually managing newspapers). Each time period is a new problem.

Figure 18.1(a) shows the shape of $F(x, D)$ for different values of D assuming, of course, that the price p is greater than the cost of the inventory c, where profits are maximized at $x = D$. Figure 18.1(b) gives a probability distribution for the random variable D, and finally figure 18.1(c) is the expected profits given that we order a quantity x, and then observe the random demand D. This figure illustrates the fundamental concave shape for the newsvendor problem, which is behind the concave shape of many resource allocation problems where we are trying to match a supply against a random demand. This behavior persists even for much more complex resource allocation problems, as long as revenues are linear in p, and costs are linear in c.

18.1.2 Two-stage resource allocation problems

In the newsvendor problem, we assume there is a single type of resource being used to satisfy a single type of demand. There are a number of settings where we have to allocate different types of resources now, after which we see the demand, and then we get to make a final decision of which resources should satisfy which demand.

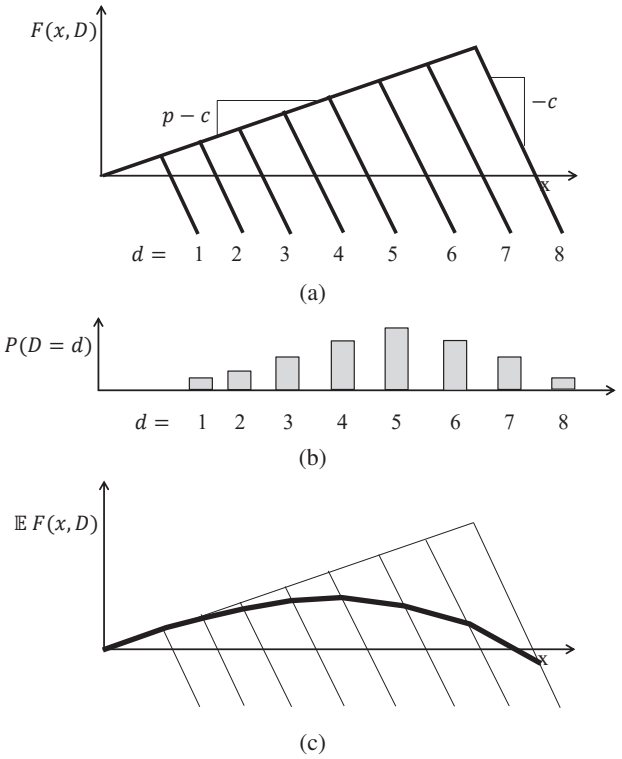


Figure 18.1 (a) The shape of the newsvendor problem for different values of the demand D; (b) The probability $P(D = d)$ of each outcome of demand; (c) The expected profits as a function of x.

EXAMPLE 18.1

An electric power utility needs to purchase expensive components that cost millions of dollars and require a year or more to order. The industry needs to maintain a supply of these in case of a failure. The problem is to determine how many units to purchase, when to purchase them, what features they should have, and where they should be stored. When a failure occurs, the company will find the closest unit that has the features required for a particular situation.

EXAMPLE 18.2

An investment bank needs to allocate funds to various investments (long-term, high risk investments, real estate, stocks, index funds, bonds, money markets, CD's). As opportunities arise, the bank will move money from one investment to another, but these transactions can take time to execute and cost money (for example, it is easiest and fastest to move money out of a money market fund).

EXAMPLE 18.3

An online bookseller prides itself in fast delivery, but this requires holding books in inventory. If orders arrive when there is no inventory, the seller may have to delay filling the order (and risk losing it) or purchase the books at a higher cost from the publisher. If the inventory is too high, the company has to choose between holding the books in inventory (tying up space and capital), discounting the book to increase sales, or selling inventory to another distributor (at a substantial discount).

EXAMPLE 18.4

An automotive manufacturer has to decide what models to design and build, and with what features. Given a three year design and build cycle, they have to create cars that will respond to an uncertain marketplace in the future. Once the models are built, customers have to adjust and purchase models that are closest to their wishes.

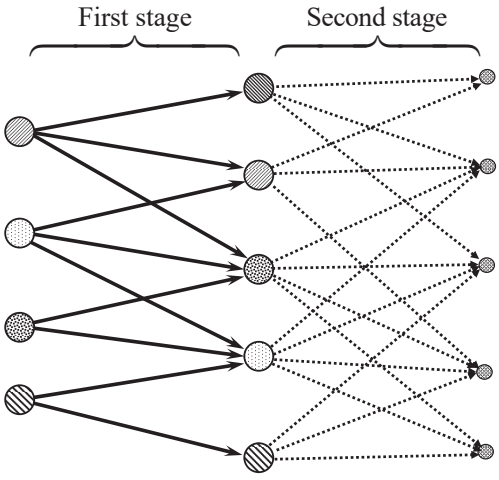


Figure 18.2 A two-stage allocation problem. The first-stage decisions have to be made before the second stage information becomes known. When this information is revealed, it is possible to re-allocate resources.

In all of these problems, we make an initial allocation decision. This could be the decision to purchase a type of equipment, build a particular model of car, or stock inventories of different types of product. Once the initial decision is made, we see information about the demand for the asset as well as the prices/costs derived from satisfying a demand (which can also be random). After this information is revealed, we may make new decisions. The goal is to make the best initial decisions given the potential downstream decisions that might be made. These problems are illustrated in figure 18.2.

This problem combines "what to do" (what type of product, where to store it) with "how much." It is a basic building block for much more complex, fully sequential resource allocation problems which we present next.

18.1.3 A general multiperiod resource allocation model*

The insights behind the newsvendor problem and the two-state resource allocation model can be leveraged into a fairly general model for dynamic resource allocation problems. In this model, we are managing a "resource" (people, equipment, blood, money) to serve "demands" (tasks, customers, jobs). We note that this model is quite general, and can be used for some fairly complex resource allocation problems.

We describe the resources and demands using:

$$\begin{aligned} R_{ta} &= \text{The number of resources with attribute } a \in \mathcal{A} \text{ in the system at time } t. \\ R_t &= (R_{ta})_{a\in\mathcal{A}}. \\ D_{tb} &= \text{The number of demands of type } b \in \mathcal{B} \text{ in the system at time } t. \\ D_t &= (D_{tb})_{b\in\mathcal{B}}. \end{aligned}$$

Both a and b are vectors of attributes of resources and demands. The state of our system is given by

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

New information is represented as exogenous changes to the resource and demand vectors, as well as to other parameters that govern the problem. These are modeled using:

$$\begin{aligned} \hat{R}_{t+1,a} &= \text{Exogenous changes to } R_{ta} \text{ from information that arrives during time interval } t \text{ (between } t \text{ and } t+1). \\ \hat{D}_{t+1,b} &= \text{Exogenous changes to } D_{tb} \text{ from information that arrives during time interval } t \text{ (between } t \text{ and } t+1). \end{aligned}$$

Our information process, then, is given by

$$W_{t+1} = (\hat{R}_{t+1}, \hat{D}_{t+1}).$$

In a blood management problem, $\hat{R}_{t+1}$ included blood donations. In a model of complex equipment such as aircraft or locomotives, $\hat{R}_{t+1}$ would also capture equipment failures or delays. In a product inventory setting, $\hat{R}_{t+1}$ could represent theft of product. $\hat{D}_{t+1}$ usually represents new customer demands, but can also represent changes to an existing demand or cancelations of orders.

Decisions are modeled using:

$$\begin{aligned} \mathcal{D}^D &= \text{Decision to satisfy a demand with attribute } b \text{ (each decision } d \in \mathcal{D}^D \text{ corresponds to a demand attribute } b_d \in \mathcal{B}). \\ \mathcal{D}^M &= \text{Decision to modify a resource (each decision } d \in \mathcal{D}^M \text{ has the effect of modifying the attributes of the resource). } \mathcal{D}^M \text{ includes the decision to "do nothing."} \\ \mathcal{D} &= \mathcal{D}^D \cup \mathcal{D}^M. \\ x_{tad} &= \text{The number of resources that initially have attribute } a \text{ that we act on with a decision of type } d \in \mathcal{D}. \\ x_t &= (x_{tad})_{a\in\mathcal{A},d\in\mathcal{D}}. \end{aligned}$$

The decisions have to satisfy constraints such as

$$\sum_{d\in\mathcal{D}} x_{tad} = R_{ta}, \tag{18.2}$$

$$\sum_{a\in\mathcal{A}} x_{tad} \leq D_{tb_d}, \quad d \in \mathcal{D}^D, \tag{18.3}$$

$$x_{tad} \geq 0. \tag{18.4}$$

We let $\mathcal{X}_t$ be the set of x_t that satisfy (18.2) - (18.4). As before, we assume that decisions are determined by a class of decision functions

$X_t^\pi(S_t)$ = a function that returns a decision vector $x_t \in \mathcal{X}_t$, where $\pi \in \Pi$ is an element of the set of functions (policies) Π.

The transition function is given generically by

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

We now have to deal with each dimension of our state variable. The most difficult, not surprisingly, is the resource vector R_t. This is handled primarily through the attribute transition function

$$a_t^x = a^{M,x}(a_t, x_t),$$

where a_t^x is the post-decision attribute (the attribute produced by action of type a before any new information has become available). For algebraic purposes, we define the indicator function

$$\delta_{a'}(a,d) = \begin{cases} 1 & \text{if } a' = a_t^x = a^{M,x}(a_t, x_t), \\ 0 & \text{otherwise.} \end{cases}$$

Using matrix notation, we can write the post-decision resource vector R_t^x using

$$R_t^x = \Delta R_t,$$

where Δ is a matrix in which $\delta_{a'}(a,d)$ is the element in row a' and column (a,d). We emphasize that the function $\delta_{a'}(a,d)$ and matrix Δ are used purely for notational convenience; in a real implementation, we just work with the transition function $a^{M,x}(a_t, d_t)$. The pre-decision resource state vector is given by

$$R_{t+1} = R_t^x + \hat{R}_{t+1}.$$

We model demands in a simple way. If a resource is assigned to a demand, then it is "served" and vanishes from the system. Otherwise, it is held to the next time period. Let

$$\begin{aligned} \delta D_{tb_d}(x) &= \text{the number of demands of type } b_d \text{ that are served at time } t, \\ &= \sum_{a\in\mathcal{A}} x_{tad} \quad d \in \mathcal{D}^D, \\ \delta D_t &= (\delta D_{tb})_{b\in\mathcal{B}}. \end{aligned}$$

The demand transition function can be written

$$\begin{aligned} D_t^x &= D_t - \delta D_t(x), \\ D_{t+1} &= D_t^x + \hat{D}_t. \end{aligned}$$

The last dimension of our model is the objective function. For our resource allocation problem, we define a contribution for each decision given by

c_{ad} = contribution earned (negative if it is a cost) from using decision d acting on resources with attribute a.

The contribution function for time period t is assumed to be linear, given by

$$C(S_t, x_t) = \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} c_{ad} x_{tad}.$$

The objective function is now given by

$$\max_{\pi \in \Pi} \mathbb{E} \left\{ \sum_{t=0}^{T} C(S_t, X_t^{\pi}(S_t)) | S_0 \right\}.$$

18.2 VALUES VERSUS MARGINAL VALUES

It is common in dynamic programming to talk about the problem of estimating the value of being in a state. When we are working on resource allocation problems, we have to make the transition from using the value of being in a state (which is extremely high dimensional for these problems) to using the *marginal* value of an additional resource R_{ta} with attribute a (if we are using our multiattribute notation). So, instead of finding a single value $V_t(R_t)$ for a high-dimensional vector R_t, we are going to compute values $\hat{v}_{ta}$ for each $a \in \mathcal{A}$ giving the marginal value of increasing R_{ta}. This means we are going to compute a vector of marginal values $\hat{v}_t = (\hat{v}_{ta})_{a \in \mathcal{A}}$ instead of a single $V_t(R_t)$.

We are going to use the context of resource allocation problems to illustrate the power of using the gradient. In principal, the challenge of estimating the slope of a function is the same as that of estimating the function itself (the slope is simply a different function). However, there can be important, practical advantages to estimating slopes. First, we may be able to approximate $V_t(R_t)$ using a linear approximation, or a piecewise linear, separable approximation.

A second and equally important difference is that if we estimate the value of being in a state, we get one estimate of the value of being in a state when we visit that state. When we estimate a gradient, we get an estimate of a derivative for *each* type of resource *all at the same time*. For example, if $R_t = (R_{ta})_{a \in \mathcal{A}}$ is our resource vector and $V_t(R_t)$ is our value function, then the gradient of the value function with respect to R_t would look like

$$\nabla_{R_t} V_t(R_t) = \begin{pmatrix} \hat{v}_{ta_1} \\ \hat{v}_{ta_2} \\ \vdots \\ \hat{v}_{ta_{|\mathcal{A}|}} \end{pmatrix},$$

where

$$\hat{v}_{ta_i} = \frac{\partial V_t(R_t)}{\partial R_{ta_i}}.$$

There may be additional work required to obtain each element of the gradient, but the incremental work can be far less than the work required to get the value function itself.

This is particularly true when the optimization problem naturally returns these gradients (for example, dual variables from a linear program), but this can even be true when we have to resort to numerical derivatives. Once we have all the calculations to solve a problem once, solving small perturbations can be relatively inexpensive.

There is one important problem class where finding the value of being in a state is equivalent to finding the derivative. That is the case of managing a single resource. In this case, the state of our system (the resource) is the attribute vector a, and we are interested in estimating the value $V(a)$ of our resource being in state a. Alternatively, we can represent the state of our system using the vector R_t, where $R_{ta} = 1$ indicates that our resource has attribute a (we assume that $\sum_{a \in \mathcal{A}} R_{ta} = 1$). In this case, the value function can be written

$$V_t(R_t) = \sum_{a \in \mathcal{A}} v_{ta} R_{ta}.$$

Here, the coefficient v_{ta} is the derivative of $V_t(R_t)$ with respect to R_{ta}.

In a typical implementation of an approximate dynamic programming algorithm, we would only estimate the value of a resource when it is in a particular state (given by the attribute vector a). This is equivalent to finding the derivative $\hat{v}_a$ only for the value of a where $R_{ta} = 1$. By contrast, computing the gradient $\nabla_{R_t} V_t(R_t)$ implicitly assumes that we are computing $\hat{v}_a$ for each $a \in \mathcal{A}$. There are some algorithmic strategies (we will describe an example of this in section 18.6) where this assumption is implicit in the algorithm. Computing $\hat{v}_a$ for all $a \in \mathcal{A}$ is reasonable if the attribute state space is not too large (for example, if a is a physical location among a set of several hundred locations). If a is a vector, then enumerating the attribute space can be prohibitive (it is, in effect, the "curse of dimensionality" revisited).

Given these issues, it is critical to first determine whether it is necessary to estimate the slope of the value function, or the value function itself. The result can have a significant impact on the algorithmic strategy.

18.3 PIECEWISE LINEAR APPROXIMATIONS FOR SCALAR FUNCTIONS

There are many problems where we have to estimate the value of having a quantity R of some resource (where R is a scalar). We might want to know the value of having R dollars in a budget, R pieces of equipment, or R units of some inventory. R may be discrete or continuous, but we are going to focus on problems where R is either discrete or is easily discretized.

Assume we have a function that is monotonically decreasing, which means that while we do not know the value function exactly, we know that $V(R+1) \leq V(R)$ (for scalar R). If our function is piecewise linear concave, then we will assume that $V(R)$ refers to the slope at R (more precisely, to the right of R). Assume our current approximation $\overline{V}^{n-1}(R)$ satisfies this property, and that at iteration n, we have a sample observation of $V(R)$ for $R = R^n$. If our function is piecewise linear concave, then $\hat{v}^n$ would be a sample realization of a derivative of the function. If we use our standard updating algorithm, we would write

$$\overline{V}^n(R^n) = (1 - \alpha_{n-1})\overline{V}^{n-1}(R^n) + \alpha_{n-1}\hat{v}^n.$$

After the update, it is quite possible that our updated approximation no longer satisfies our monotonicity property. Below we review two strategies for maintaining monotonicity:

The leveling algorithm - A simple method that imposes monotonicity by simply forcing elements of the series which violate monotonicity to a larger or smaller value so that monotonicity is restored.

The CAVE algorithm - If there is a monotonicity violation after an update, CAVE simply expands the range of the function over which the update is applied.

The leveling algorithm

The leveling algorithm uses a simple updating logic that can be written as follows:

$$\overline{V}^n(y) = \begin{cases} (1-\alpha_{n-1})\overline{V}^{n-1}(R^n) + \alpha_{n-1}\hat{v}^n & \text{if } y = R^n, \\ \overline{V}^n(y) \vee \left\{(1-\alpha_{n-1})\overline{V}^{n-1}(R^n) + \alpha_{n-1}\hat{v}^n\right\} & \text{if } y > R^n, \\ \overline{V}^n(y) \wedge \left\{(1-\alpha_{n-1})\overline{V}^{n-1}(R^n) + \alpha_{n-1}\hat{v}^n\right\} & \text{if } y < R^n, \end{cases} \tag{18.5}$$

where $x \wedge y = \max\{x, y\}$, and $x \vee y = \min\{x, y\}$. Equation (18.5) starts by updating the slope $\overline{V}^n(y)$ for $y = R^n$. We then want to make sure that the slopes are declining. So, if we find a slope to the right that is larger, we simply bring it down to our estimated slope for $y = R^n$. Similarly, if there is a slope to the left that is smaller, we simply raise it to the slope for $y = R^n$. The steps are illustrated in figure 18.3.

The CAVE algorithm

A particularly useful variation is to perform an initial update (when we compute $\bar{y}$) over a wider interval than just $y = R^n$. Assume we are given a parameter δ^0 which has been chosen so that it is approximately 20 to 50 percent of the maximum value that R^n might take. Now compute $\overline{V}^n(y)$ using

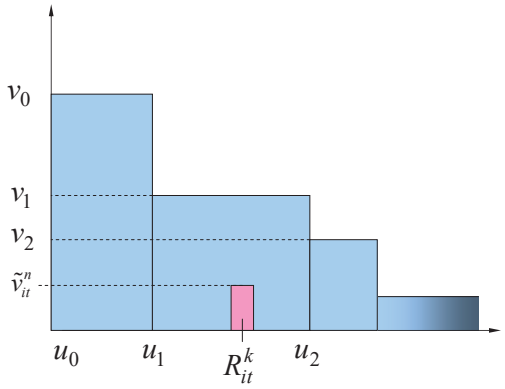
$$\overline{V}^n(y) = \begin{cases} (1-\alpha_{n-1})\overline{V}^{n-1}(y) + \alpha_{n-1}\hat{v}^n, & R^n - \delta^n \leq y \leq R^n + \delta^n, \\ \overline{V}^{n-1}(y) & \text{otherwise.} \end{cases}$$

Here, we are using $\hat{v}^n$ to update a wider range of the interval. We then apply the same logic for maintaining monotonicity (concavity if these are slopes). We start with the interval $R^n \pm \delta^0$, but we have to periodically reduce δ^0. We might, for example, track the objective function (call it F^n), and update the range using

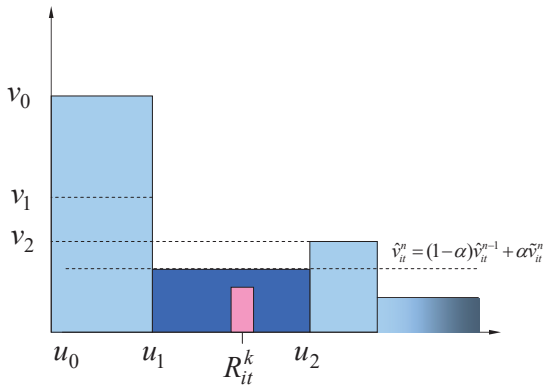
$$\delta^n = \begin{cases} \delta^{n-1} & \text{If } F^n \geq F^{n-1} - \epsilon, \\ \max\{1, .5\delta^{n-1}\} & \text{otherwise.} \end{cases}$$

While the rules for reducing δ^n are generally ad hoc, we have found that this is critical for fast convergence. The key is that we have to pick δ^0 so that it plays a critical scaling role, since it has to be set so that it is roughly on the order of the maximum value that R^n can take.

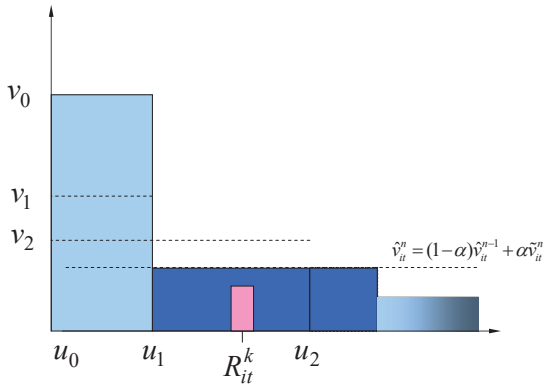
The CAVE algorithm, properly tuned, is likely to be the better of the two methods, but tuning is important and introduces an additional step. We suggest using CAVE if you anticipate that you are going to be doing quite a bit of work with a particular problem class.



18.3a: Initial monotone function.



18.3b: After update of a single segment.



18.3c: After leveling operation.

Figure 18.3 Steps of the leveling algorithm. Figure 18.3a shows the initial monotone function, with the observed R and observed value of the function $\hat{v}$. Figure 18.3b shows the function after updating the single segment, producing a non-monotone function. Figure 18.3c shows the function after monotonicity restored by leveling the function.

18.4 REGRESSION METHODS

As in chapter 3 we can create regression models where the are manipulations of the number of resources of each type. For example, we might use

$$\overline{V}(R) = \theta_0 + \sum_{a \in \mathcal{A}} \theta_{1a} R_a + \sum_{a \in \mathcal{A}} \theta_{2a} R_a^2, \tag{18.6}$$

where $\theta = (\theta_0, (\theta_{1r})_{r \in \mathcal{R}}, (\theta_{2r})_{r \in \mathcal{R}})$ is a vector of parameters that are to be determined. The choice of explanatory terms in our approximation will generally reflect an understanding of the properties of our problem. For example, equation (18.6) assumes that we can use a mixture of linear and separable quadratic terms. A more general representation is to assume that we have developed a family $\mathcal{F}$ of basis functions $(\phi_f(R))_{f \in \mathcal{F}}$. Examples of a basis function are

$$\begin{aligned}
\phi_f(R) &= R_{a_f}^2, \\
\phi_f(R) &= \left(\sum_{a \in \mathcal{A}_f} R_a \right)^2 \quad \text{for some subset } \mathcal{R}_f, \\
\phi_f(R) &= (R_{a_1} - R_{a_2})^2, \\
\phi_f(R) &= |R_{a_1} - R_{a_2}|.
\end{aligned}$$

A common strategy is to capture the number of resources at some level of aggregation. For example, if we are purchasing emergency equipment, we may care about how many pieces we have in each region of the country, and we may also care about how many pieces of a type of equipment we have (regardless of location). These issues can be captured using a family of aggregation functions $G_f,\ f \in \mathcal{F}$, where $G_f(a)$ aggregates an attribute vector a into a space $\mathcal{R}^{(f)}$ where for every basis function f there is an element $a_f \in \mathcal{R}^{(f)}$. Our basis function might then be expressed using

$$\phi_f(R) = \sum_{a \in \mathcal{A}} \mathbb{1}_{\{G_f(a) = a_f\}} R_a.$$

We have written our basis functions purely in terms of the resource vector, but it is possible for them to be written in terms of other parameters in a more complex state vector, such as asset prices.

Given a set of basis functions, we can write our value function approximation as

$$\overline{V}(R|\theta) = \sum_{f \in \mathcal{F}} \theta_f \phi_f(R). \tag{18.7}$$

It is important to keep in mind that $\overline{V}(R|\theta)$ (or more generally, $\overline{V}(S|\theta)$), is any functional form that approximates the value function as a function of the state vector parameterized by θ. Equation (18.7) is a classic linear-in-the-parameters function. We are not constrained to this form, but it is the simplest and offers some algorithmic shortcuts.

The issues that we encounter in formulating and estimating $\overline{V}(R|\theta)$ are the same that any student of statistical regression would face when modeling a complex problem. The major difference is that our data arrives over time (iterations), and we have to update our formulas recursively. Also, it is typically the case that our observations are nonstationary.

This is particularly true when an update of a value function depends on an approximation of the value function in the future (as occurs with value iteration or any of the TD(λ) classes of algorithms). When we are estimating parameters from nonstationary data, we do not want to equally weight all observations.

The problem of finding θ can be posed in terms of solving the following stochastic optimization problem

$$\min_{\theta} \mathbb{E}\frac{1}{2}(\overline{V}(R|\theta) - \hat{V})^2.$$

We can solve this using a stochastic gradient algorithm, which produces updates of the form

$$\begin{aligned}
\bar{\theta}^n &= \bar{\theta}^{n-1} - \alpha_{n-1}(\overline{V}(R^n|\bar{\theta}^{n-1}) - \hat{V}(\omega^n))\nabla_\theta \overline{V}(R^n|\theta^n) \\
&= \bar{\theta}^{n-1} - \alpha_{n-1}(\overline{V}(R^n|\bar{\theta}^{n-1}) - \hat{V}(\omega^n)) \begin{pmatrix} \phi_1(R^n) \\ \phi_2(R^n) \\ \vdots \\ \phi_F(R^n) \end{pmatrix}.
\end{aligned}$$

If our value function is linear in R_t, we would write

$$\overline{V}(R|\theta) = \sum_{a\in\mathcal{A}} \theta_a R_a.$$

In this case, our number of parameters has shrunk from the number of possible realizations of the entire vector R_t to the size of the attribute space (which, for some problems, can still be large, but nowhere near as large as the original state space). For this problem, $\phi(R^n) = R^n$.

It is not necessarily the case that we will always want to use a linear-in-the-parameters model. We may consider a model where the value increases with the number of resources, but at a declining rate that we do not know. Such a model could be captured with the representation

$$\overline{V}(R|\theta) = \sum_{a\in\mathcal{A}} \theta_{1a} R_a^{\theta_{2a}},$$

where we expect $\theta_2 < 1$ to produce a concave function. Now, our updating formula will look like

$$\begin{aligned}
\theta_1^n &= \theta_1^{n-1} - \alpha_{n-1}(\overline{V}(R^n|\bar{\theta}^{n-1}) - \hat{V}(\omega^n))(R^n)^{\theta_2}, \\
\theta_2^n &= \theta_2^{n-1} - \alpha_{n-1}(\overline{V}(R^n|\bar{\theta}^{n-1}) - \hat{V}(\omega^n))(R^n)^{\theta_2} \ln R^n
\end{aligned}$$

where we assume the exponentiation operator in $(R^n)^{\theta_2}$ is performed componentwise.

We can put this updating strategy in terms of temporal differencing. As before, the temporal difference is given by

$$\delta_\tau = C_\tau(R_\tau, x_{\tau+1}) + \overline{V}_{\tau+1}^{n-1}(R_{\tau+1}) - \overline{V}_{\tau}^{n-1}(R_\tau).$$

The original parameter updating formula (equation (16.7)) when we had one parameter per state now becomes

$$\bar{\theta}^n = \bar{\theta}_t^{n-1} + \alpha_{n-1}\sum_{\tau=t}^{T} \lambda^{\tau-t}\delta_\tau \nabla_\theta \overline{V}(R^n|\bar{\theta}^n).$$

It is important to note that in contrast with most of our other applications of stochastic gradients, updating the parameter vector using gradients of the objective function requires mixing the units of θ with the units of the value function. In these applications, the stepsize α_{n-1} has to also perform a scaling role.

18.5 SEPARABLE PIECEWISE LINEAR APPROXIMATIONS

Scalar, piecewise linear functions have proven to be an exceptionally powerful way of solving high dimensional stochastic resource allocation problems. We can describe the algorithm with a minimum of technical details using what is known as a "plant-warehouse-customer" model, which we presented in section 18.1.2. Imagine that we have the problem depicted in figure 18.4a. We start by shipping "product" out of the four "plant" nodes on the left, and we have to decide how much to send to each of the five "warehouse" nodes in the middle. After making this decision, we then observe the demands at the five "customer" nodes on the right.

We can solve this problem using separable, piecewise linear value function approximations. Assume we have an initial estimate of a piecewise linear value function for resources at the warehouses (setting these equal to zero is fine). This gives us the network shown in figure 18.4b, which is a small linear program, even when we have hundreds (or thousands) of plant and warehouse nodes. Solving this problem gives us a solution of how much to send to each node.

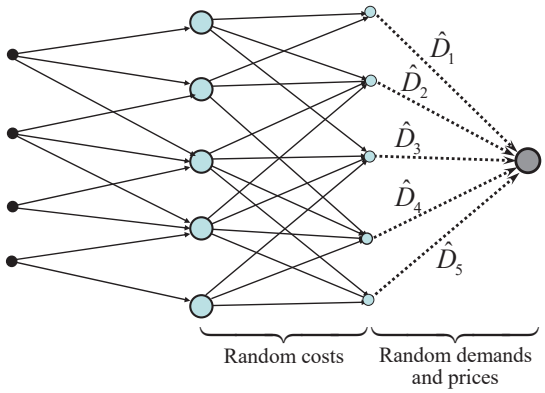
We then use the solution to the first stage (which gives us the resources available at each warehouse node), take a Monte Carlo sample of each of the demands, and solve a second linear program that sends product from each warehouse to each customer. What we want from this stage is the dual variable for each warehouse node, which gives us an estimate of the marginal value of resources at each node. Note that some care needs to be used here, because these dual variables are not actually estimates of the value of one more resources, but rather are subgradients, which means that they may be the value of the last resource or the next resource, or something in between.

Finally, we use these dual variables to update the piecewise linear value functions using the methods described above. This process is repeated until the solution no longer seems to be improving.

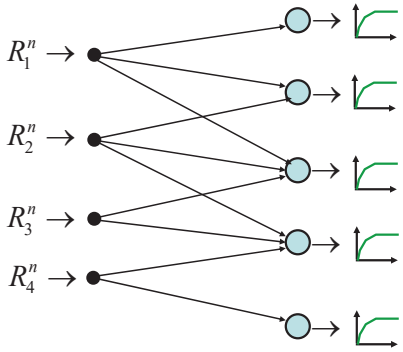
Although we have described this algorithm in the context of a two-stage problem, the same basic strategy can be applied for problems with many time periods. Using approximate value iteration (TD(0)), we would step forward in time, and after solving each linear program we would stop and use the dual variables for the constraints (18.2) to update the value functions from the previous time period (more specifically, around the previous post-decision state). For a finite horizon problem, we would proceed until the last time period, then repeat the entire process until the solution seems to be converging.

With more work, we can implement a backward pass (TD(1)) by avoiding any value function updates until we reach the final time period, but we would have to retain information about the effect of incrementing the resources at each node by one unit (this is best done with a numerical derivative). We would then need to step back in time, computing the marginal value of one more resource at time t using information about the value of one more resource at time $t + 1$. These marginal values would be used to update the value function approximations.

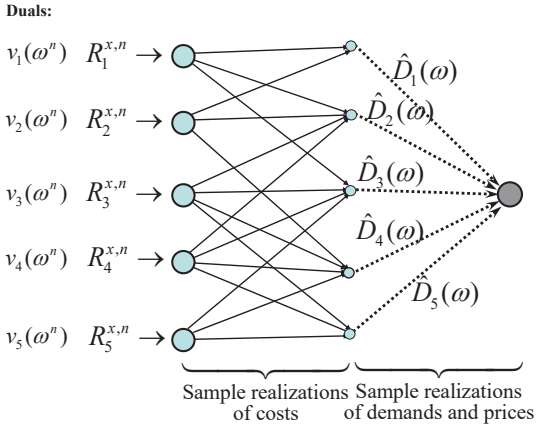
This algorithmic strategy has some nice features:



18.4a: The two-stage problem with stochastic second-stage data.



18.4b: Solving the first stage using a separable, piecewise linear approximation of the second stage.



18.4c: Solving a Monte Carlo realization of the second stage and obtaining dual variables.

Figure 18.4 Steps in estimating separable, piecewise-linear approximations for two-stage stochastic programs.

- This is a very general model with applications that span equipment, people, product, money, energy and vaccines. It is ideally suited for "single layer" resource allocation problems (one type of resource, rather than pairs such as pilots and aircraft, blood and patients, or trucks and deliveries), although many two-layer problems can be reasonably approximated as single-layer problems.
- The methodology scales to very large problems, with hundreds or thousands of nodes, and tens of thousands of dimensions in the decision vector.
- We do not need to solve the exploration-exploitation problem. A pure exploitation strategy works fine. The reason has to do with the concavity of the value function approximations, which has the effect of pushing suboptimal value functions toward the correct solution.
- Piecewise linear value function approximations are quite robust, and avoid making any simplifying assumptions about the shapes of the value functions.

18.6 BENDERS DECOMPOSITION FOR NONSEPARABLE APPROXIMATIONS**

While the use of separable, piecewise linear approximations has proven effective (especially for discrete problems where flows need to be integer), the use of a separable approximation will inevitably introduce errors. It is possible to create a nonseparable approximation using an approach called *Benders decomposition* which approximates the value function by minimizing over a set of linear hyperplanes, known as cutting planes.

We are going to begin by presenting the idea of Benders decomposition for a simple two-stage resource allocation problem.

18.6.1 Benders' decomposition for two-stage problems

Cutting planes represent a powerful strategy for representing concave (or convex if we are minimizing), piecewise-linear functions for multidimensional problems. This method evolved originally as a technique in the 1970s for solving complex integer programs which benefited from separating decision variables into two classes (say, optimizing warehouse locations, and then allocating demands to warehouses). The method was then adapted to the types of sequential decision problems in the early 1990s that arise in two-stage and multistage stochastic resource allocation problem.

Historically, dynamic programming has been viewed as a technique for small, discrete optimization problems, while stochastic programming has been the field that handles uncertainty within math programs (which are typically characterized by high-dimensional decision vectors and large numbers of constraints). The connections between stochastic programming and dynamic programming, historically viewed as diametrically competing frameworks, have been largely overlooked. This section is designed to bridge the gap between stochastic programming and approximate dynamic programming. Our presentation is facilitated by notational decisions (such as our use of x_t for decisions), and our use of the post-decision state variable, which eliminates the expectation from within the maximization problem for each period.

In this section, we are going to put our sampling in the context of an iterative algorithm, where we choose sample ω^n at the n^{th} iteration. This contrasts with our previous style

of choosing a fixed sample $w_1, \ldots, w_K$. We just want to emphasize that the change in notation reflects a change in the context of how sampling is done. We do this because we may have to choose a sample $w_1^n, \ldots, w_K^n$ at the n^{th} iteration.

For example, let R_t be the vector of inventories of product at each of the fulfillment centers, and let x_t be the replenishment decisions that will arrive at time $t+1$. The decisions x_t have to satisfy a set of constraints that we represent generically as

$$A_t x_t = R_t.$$

These inventories have to be used to satisfy random demands D_{t+1} (which has the same dimensionality as R_t). The inventories R_{t+1} are then given by

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

where x_t is the vector of flows from one facility to the next, and the matrix B_t sums the flows into each facility. Here, we have added in some noise, $\hat{R}_{t+1}$, that might account for damaged or delayed shipments. We would also have observed the demands D_{t+1}, and updated transportation costs c_{t+1}, which are random because of the need to move by for-hire trucking companies. We also note that the matrices A_t and B_t capture travel times; if these are random, then at time t the matrices A_{t+1} and B_{t+1} are also random.

This means the information that is revealed by time $t+1$ is

$$W_{t+1} = (A_{t+1}, B_{t+1}, c_{t+1}, D_{t+1}, \hat{R}_{t+1}),$$

which in turn gives us our (pre-decision) state at time $t+1$ as

$$S_{t+1} = (R_{t+1}, A_{t+1}, B_{t+1}, c_{t+1}, D_{t+1}).$$

We are going to simplify our presentation by assuming that A_{t+1}, B_{t+1}, c_{t+1} and D_{t+1} are independent of any previous information (a property known as *interstage independence* in the stochastic programming literature), which means that our post-decision state is

$$S_t^x = R_t^x = B_t x_t.$$

Since S_t^x is determined by x_t, the stochastic programming literature writes this state variable as x_t which, while mathematically accurate, is of much higher dimensionality than R_t^x (which could be a scalar if we have a single warehouse that holds inventories). Either representation works fine for what we are going to do.

If we use the pre-decision state, the problem to find x_t at time t is given by

$$\max_{x_t} \big(c_t x_t + \mathbb{E}_{W_{t+1}} V_{t+1}(R_{t+1}, W_{t+1})\big). \tag{18.8}$$

Note that the information vector W_{t+1} is extremely high dimensional, which would complicate both taking the expectation $\mathbb{E}_{W_{t+1}}$ as well as approximating $V_{t+1}(R_{t+1}, W_{t+1})$. But if we use the post-decision state, we get the much simpler problem

$$\max_{x_t} \big(c_t x_t + V_t^x(R_t^x)\big). \tag{18.9}$$

This problem is solved subject to the constraints,

$$A_t x_t = R_t, \tag{18.10}$$

$$x_t \geq 0, \tag{18.11}$$

where (18.10) represents constraints on how much inventory can be placed in each fulfillment center (captured by R_t).

We then solve the second stage problem (at time $t+1$) to determine x_{t+1}, given the first stage decisions. Assume that we observe outcome ω for the random variable W. We get to see the new information $W_{t+1}(\omega)$ before we compute x_{t+1}, so we capture this by writing $x_{t+1}(\omega)$. The resulting problem would be written

$$V_{t+1}(x_t, W_{t+1}(\omega)) = \max_{x_{t+1}(\omega)} c_{t+1}(\omega)x_{t+1}(\omega), \tag{18.12}$$

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

$$\begin{aligned} A_{t+1}(\omega)x_{t+1}(\omega) &\leq R_{t+1}(\omega), & (18.13)\\ B_{t+1}(\omega)x_{t+1}(\omega) &\leq D_{t+1}(\omega), & (18.14)\\ x_{t+1}(\omega) &\geq 0. & (18.15) \end{aligned}$$

Equation (18.13) imposes flow conservation on the flows of inventories. Equation (18.14) represents the demand constraints, where we assume our contribution vector c_{t+1} is designed to give a high incentive to meet demand. Let $\beta_{t+1}(\omega)$ be the dual variable of the resource constraint (18.13) which reflects the effect of the time t decision x_t on time period $t+1$.

Our strategy will be to replace $V_t^x(x_t)$ with an approximation that is created by generating a series of hyperplanes, and then taking the minimum across these hyperplanes as our approximation. This "approximation" will, in the limit, produce an exact representation of $V_t^x(x_t)$.

The value function $V_{t+1}(x_t, W_{t+1})$ is known in the stochastic programming literature as the *recourse function* since it allows us to respond to different outcomes using the *recourse variables* $x_{t+1}(\omega)$ which are chosen after choosing x_t and observing $W_{t+1}(\omega)$. Thus, we might want to satisfy demand in Texas from a nearby fulfillment center in Houston, but if that center does not have sufficient inventory, our *recourse* is to satisfy demand from a more distant center in Chicago.

We face the challenge of approximating the function $V_{t+1}(x_t) = \mathbb{E}V_{t+1}(x_t, W_{t+1})$ so that we can solve the initial problem for x_t in equation (18.9). It would also be nice if we could do this in a way so that we can solve the first stage problem as a linear program, which makes it easy to handle the vector x_t. There are several strategies we can draw on, but here we are going to illustrate a powerful idea known as Benders decomposition. In a nutshell, our second stage function $V_{t+1}(x_t, W_{t+1})$ is a linear program, which means that it is concave in the right hand side constraint $B_1 x_0$ (because we are maximizing).

We illustrate Benders decomposition in the context of solving a sampled version of the problem. We do this by replacing our original full sample space Ω (over which the original expectation $\mathbb{E}$ is defined) with a sampled set of outcomes $\mathcal{W} = (\omega^1, \ldots, \omega^N)$. For each solution, we would obtain the optimal value $\hat{V}_{t+1}(x_t, w)$, and the corresponding dual variable $\beta(w)$ for $w \in \mathcal{W}$. We then average over the outcomes to create an approximation of the post-decision value function $V_t^x(x_t)$ which we denote $\overline{V}_t^x(x_t)$, given by

$$\overline{V}_t^x(x_t) = \frac{1}{N}\sum_{n=1}^{N} \hat{V}_{t+1}(x_t, \omega^n).$$

Benders decomposition iteratively builds up an approximation of $V_{t+1}(x_t)$ by constructing a series of *supporting hyperplanes* (see figure 18.5) derived by solving the second stage

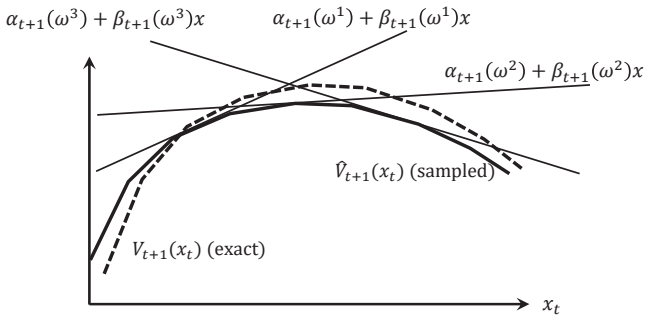


Figure 18.5 Illustration of Benders cuts shown next to exact ($V_{t+1}(x_t)$) and sampled ($\hat{V}_{t+1}(x_t)$) recourse functions.

linear program for individual samples w of the random vector W_{t+1}. We do this by solving equations (18.12)-(18.15) for the sample realizations $\Omega = \{\omega^n, n = 1, \ldots, N\}$, and obtain

$$\begin{aligned} \alpha^n_{t+1}(\omega^n) &= V_{t+1}(x_t, W_{t+1}(\omega^n)), \\ \beta^n_{t+1} &= \beta_{t+1}(\omega^n). \end{aligned}$$

where $\beta_{t+1}(\omega^n)$ is the dual variable for constraint (18.13). We then solve

$$x^*_t = \arg\max_{x_t, z_t} \big(c_t x_t + z_t), \tag{18.16}$$

subject to (18.10) - (18.11) and

$$z_t \leq \alpha^n_{t+1}(\omega^n) + \beta^n_{t+1}(\omega^n)x_t, \quad n = 1, \ldots, N. \tag{18.17}$$

Equation (18.17) creates a multidimensional envelop, depicted in figure 18.5, which depicts the sampled function $\hat{V}_{t+1}(x_t)$ and the original true function $V_{t+1}(x_t)$. Note that the hyperplanes touch the sampled function $\hat{V}_{t+1}(x_t)$, but only approximate the true function $V_{t+1}(x_t)$.

Our indexing of time deserves a bit of explanation. The coefficients $\alpha^n_{t+1}(\omega^n)$ and $\beta^n_{t+1}(\omega^n)$ are indexed by $t + 1$ because they depend on the specific sampled observation $W_{t+1}(\omega)$ of the new information that becomes known by $t + 1$. However, z_t works like an expectation; equation 18.17 is taking the minimum across all these cuts, creating z_t which does not depend on a single realization ω.

The steps of the algorithm implementing this method are shown in figure 18.6.

We close by noting that this is one way of solving convex problems, but it requires assuming that the sampled approximation will provide a good solution. This has opened a body of literature focusing on the design of good samples, which is challenging in the high dimensional settings of linear programs.

It would be easy to conclude that using multidimensional Benders cuts would be better than using separable, piecewise linear approximations. The separable, piecewise linear approximations are particularly useful when managing discrete resources (trucks, locomotives) since it is much easier to obtain integer solutions when using a piecewise linear approximation where the kinks occur on integer values of R_{ta}.

We compared the two approaches in the setting of managing energy for a fleet of batteries connected on the grid, where the amount of energy being stored in each battery is continuous.

Step 0. Initialization:

Step 0a. Initialize V_t^0

Step 0c. Set $n = 1$.

Step 1. Solve

$$x_t^n = \arg\max_{x_t, z_t} \big(c_t x_t + z_t\big),$$

subject to

$$z_t \leq \alpha_{t+1}^m(\omega^m) + \beta_{t+1}^m(\omega^m)x_t, \;\; m = 1, \ldots, n-1.$$

Step 2. For $k = 1, \ldots, K$:

$$\hat{V}_{t+1}(x_t^n, W_{t+1}(\omega^k)) = \max_{x_{t+1}(\omega^k)} c_{t+1}(\omega^k)x_{t+1}(\omega^k),$$

subject to (18.13)-(18.15). Obtain dual $\beta_{t+1}^n(\omega^k)$ for equation (18.14) for each ω^k.

Step 3. Compute:

$$\begin{aligned} \alpha_t^n &= \frac{1}{K}\sum_{k=1}^{K} \hat{V}_{t+1}(x_t^n, \omega^k), \\ \beta_t^n &= \frac{1}{K}\sum_{k=1}^{K} \beta_{t+1}^n(\omega^k). \end{aligned}$$

Step 4. Increment n. If $n \leq N$ go to Step 1.

Step 5. Return solution x_t^N.

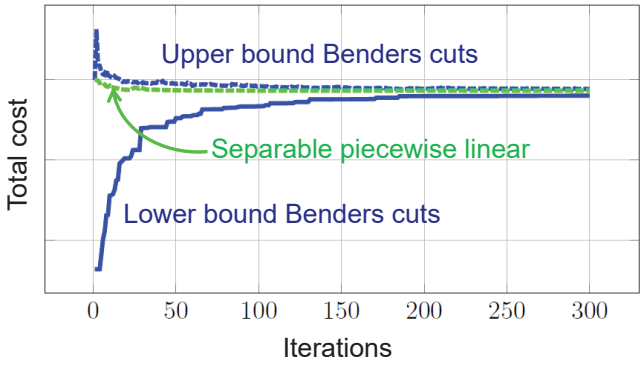
Figure 18.6 Illustration of Benders decomposition for two-stage stochastic optimization using sampled model.

Also, because it is easy to move energy between any pair of locations on the grid, we would expect the problem to be highly nonseparable. Figure 18.7 shows the performance of Benders cuts (see the upper bound performance) against separable, piecewise linear value function approximations for grids with 25 batteries (a) and 50 batteries (b).

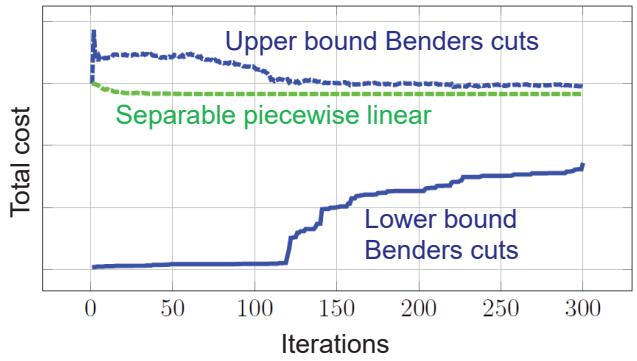
The results show that the SPWL approximation has slightly faster convergence for the case with 25 batteries. For the grid with 50 batteries, the separable approximation seems to show much faster convergence. We suspect a reason that the separable approximation works so well is that the updates are more efficient; Benders cuts in high dimensions are less efficient because the cuts do not contribute to the quality of the marginal value of *each* battery, whereas this is not the case with the separable approximations, where each VFA is updated every iteration.

18.6.2 Asymptotic analysis of Benders with regularization**

The previous section described the basic idea of Benders decomposition using a fixed sample to represent the uncertainty of the second stage. Here, we present an asymptotic version of Benders that is in the theme of the other iterative algorithms presented in this chapter. This version was first introduced as *stochastic decomposition*. We begin by introducing the basic algorithm, followed by a variant known as regularization that has been found to stabilize performance.



18.7(a) Benders (upper bound) vs. separable, piecewise linear for grid with 25 batteries.



18.7(b) Benders (upper bound) vs. separable, piecewise linear for grid with 50 batteries.

Figure 18.7 Comparison of Benders cuts vs. separable, piecewise linear value function approximations for allocating energy between a fleet of batteries over a power grid.

The basic algorithm We begin by presenting the two-stage stochastic programming model we first presented in section 18.6:

$$\max_{x_0} \big(c_0 x_0 + \mathbb{E}Q_1(x_0, W)\big), \tag{18.18}$$

subject to

$$A_0 x_0 = b, \tag{18.19}$$

$$x_0 \geq 0. \tag{18.20}$$

We are going to again solve the original problem (18.18) using a series of Benders cuts, but this time we are going to construct them somewhat different. The approximated problem still looks like

$$x^n = \arg\max_{x_0, z} (c_0 x_0 + z), \tag{18.21}$$

subject to (18.19)-(18.20) and

$$z \leq \alpha_m^n + \beta_m^n x_0, \quad m = 1, \ldots, n-1. \tag{18.22}$$

Of course, for iteration $n = 1$ we do not have any cuts.

The second stage problem which is solved for a given value $W(\omega)$ which specifies the costs and the demand D_1. In our iterative algorithm, we solve the problem for ω^n, using the solution x_0^n from the first stage:

$$Q_1(x_0^n, \omega^n) = \max_{x_1(\omega^n)} c_1(\omega^n)x_1(\omega^n), \tag{18.23}$$

subject to:

$$\begin{aligned} A_1x_1(\omega^n) &\leq B_1x_0^n, & (18.24)\\ B_1x_1(\omega^n) &\leq D_1(\omega^n), & (18.25)\\ x_1(\omega^n) &\geq 0. & (18.26) \end{aligned}$$

As before, we let $\hat{\beta}^n$ be the dual variable for the resource constraint (18.24) when we solve the problem using sample ω^n. Then let

$$\begin{aligned} \alpha_n^n &= \frac{1}{n}\sum_{m=1}^{n} Q_1(x_0^m, \omega^m),\\ \beta_n^n &= \frac{1}{n}\sum_{m=1}^{n} \hat{\beta}^m. \end{aligned}$$

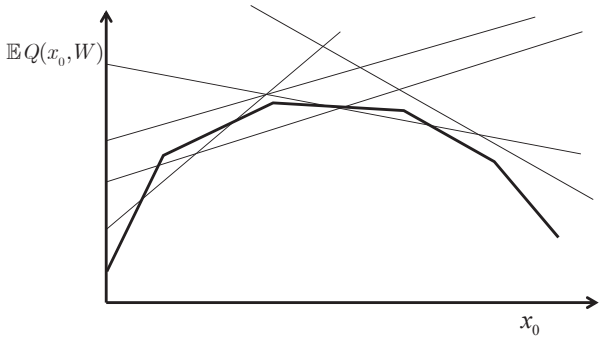
Thus, we compute α_n^n by averaging all the prior objective functions for the second stage, and then we compute β_n^n by averaging all the prior dual variables. We finally update all prior α_m^n and β_m^n for $m < n$ using

$$\begin{aligned} \alpha_m^n &= \frac{n-1}{n}\alpha_m^{n-1}, \ \ m = 1, \ldots, n-1,\\ \beta_m^n &= \frac{n-1}{n}\beta_m^{n-1}, \ \ m = 1, \ldots, n-1. \end{aligned}$$

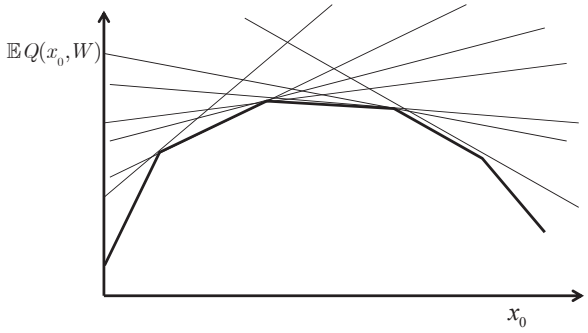
Aside from the differences in how the Benders cuts are computed, the major difference between this implementation and our earlier sampled solution given in section 18.6 is that in this recursive formulation, the samples ω are drawn from the full sample space Ω rather than a sampled one. When we solve the sampled version of the problem, we solve it exactly in a finite number of iterations, but we only obtain an optimal solution to a sampled problem. Here, we have an algorithm that will asymptotically converge to the optimal solution of the original problem.

Figure 18.8 illustrates the cuts generated using stochastic decomposition. It is useful to compare the cuts generated using stochastic decomposition to those generated when we used a sampled version of the problem in section 18.6 as depicted in figure 18.5. When we were solving our sampled version, we could compute the expectation exactly, which is why the cuts were tight. Here, we are sampling from the full probability space, and as a result we get cuts that approximate the function, but nothing more. However, in the limit as $n \to \infty$, the cuts will converge to the true function in the vicinity of the optimum.

Which is better? Hard to say. While it is nice to have an algorithm that is asymptotically optimal, we can only run a finite number of iterations. The sampled problem will be more stable due to the averaging that takes place in every iteration, but we then have to solve a linear program for every ω in the sampled problem, a step that involves much more computational overhead than the recursive version.



(a) Benders cuts in the early iterations



(b) Benders cuts in the limit

Figure 18.8 Illustration of cuts generated using stochastic decomposition (a) in the early iterations and (b) in the limit.

18.6.3 Benders with regularization

Regularization is a tool that will come up repeatedly when estimating functions from data. The same is true with Benders decomposition. Regularization is handled through a minor modification of the approximate optimization problem (18.21) which becomes

$$x^n = \arg\max_x \left(c_0 x_0 + z + \rho^n (x - \bar{x}^{n-1})^2\right), \tag{18.27}$$

which is solved subject to (18.19)-(18.20) and the Benders cut constraints (18.22). The parameter ρ^n is a decreasing sequence that needs to be scaled to handle the difference in the units between the costs and the term $(x - \bar{x}^{n-1})^2$. $\bar{x}^{n-1}$ is the regularization term which is updated each iteration; the idea with regularization is to keep x^n from straying too far from a previous solution, especially in the early iterations.

The use of the squared deviation $(x - \bar{x}^{n-1})^2$ is known as L_2 regularization, which might be written as $\|x - \bar{x}^{n-1}\|_2^2$. An alternative is L_1 regularization which minimizes the absolute value of the deviation, which would be written as $|x - \bar{x}^{n-1}|$.

There are different ways of setting the regularization term, but the simplest one just uses $\bar{x}^{n-1} = x^{n-1}$. Other ideas involve smoothing several previous iterations. The

regularization coefficient is any declining sequence such as

$$\rho^k = r\rho^{k-1}$$

for some factor $r < 1$, starting with an initial ρ^0 that has to be chosen to handle the scaling.

Properly implemented, regularization offers not only theoretical guarantees, but has also been found to accelerate convergence and stabilize the performance of the algorithm.

18.7 LINEAR APPROXIMATIONS FOR HIGH-DIMENSIONAL APPLICATIONS

Imagine now that we are going to use basis functions that are linear in the resource variables (not to be confused with linear models that are linear in the parameters). A linear approximation for our value function approximation is given by

$$\overline{V}^n(R_t) = \sum_{a \in \mathcal{A}} \bar{v}^n_{ta} R_{ta},$$

where $\bar{v}^n_{ta}$ is our estimate of the marginal value of resources with attribute a after n iterations. We can estimate these slopes by using the methods described above to obtain $\hat{v}^n_{ta}$ which is our sampled estimate of the marginal value of R_{ta} at iteration n. We then update our estimate of the linear approximation using

$$\bar{v}^n_{ta} = (1 - \alpha_n)\bar{v}^{n-1}_{ta} + \alpha_n \hat{v}^n_{ta},$$

where α_n is our stepsize (see chapter 6) which might depend on how many times we have observed $\hat{v}^n_a$ for attribute a.

Linear value function approximations can be particularly useful for high-dimensional problems where the space of attributes $\mathcal{A}$ is so large that the elements of R_{ta} are likely to be quite small (mostly zero, sometimes 1). However, this creates the problem that we may have trouble computing $\hat{v}^n_{ta}$ for every $a \in \mathcal{A}$.

Consider, for example, the problem of assigning truck drivers to loads, where a is the attributes of the truck driver. The attribute vector a might include

a_1 = current location of the driver (or location where he is headed),
a_2 = the location of where he lives,
a_3 = how many days he has been driving since he was last at home,
a_4 = the type of truck trailer he is pulling (for example, a dry van or refrigerated trailer),
a_5 = the driver's nationality.

The size of this attribute space could easily be in the millions, while we may be optimizing a fleet of 100 or 1000 drivers. Computing $\hat{v}^n_{ta}$, which might require reoptimizing the time t optimization problem to get the marginal value of R_{ta} for each $a \in \mathcal{A}$, would be computationally expensive.

The way to circumvent this "curse of dimensionality" is to use the power of hierarchical aggregation introduced in section 3.6.1. The idea is to create a family of reduced attribute vectors $a^{(g)}$ where $a^{(0)}$ is the original full attribute vector. Then, create a series of more compact vectors $a^{(1)}, a^{(2)}, \ldots, a^{(G)}$, where $a^{(G)}$ might have a single attribute with a small

number of values. Hierarchical aggregation maintains a set of weights $w_a^{(g),n}$ that depend on the level of aggregation, as well as the attribute vector a, where

$$\sum_{g=0}^{G} w_a^{(g),n} = 1.$$

These weights are given by one over the variance plus square of the bias, normalized so they sum to 1 (see section 3.6.1 for details).

We then maintain a value of averaged estimates $\bar{v}_a^{(g),n}$, recognizing that we may not have any observations for some aggregation levels g for some (in fact many) attribute vectors a. When this is the case, we set $w_a^{(g),n} = 0$. Finally, we obtain our estimate of the marginal value of R_{ta} using

$$\bar{v}_a^n = \sum_{g=0}^{G} w_a^{(g),n} \bar{v}_a^{(g),n}.$$

18.8 RESOURCE ALLOCATION WITH EXOGENOUS INFORMATION STATE

The methods in this chapter have demonstrated how we can find effective value function approximations for resource allocation problems, even when the resource vector R_t is very high-dimensional. All of this work assumed that R_t was the *only* state variable. The message here is simple: high-dimensional problems are not hard, as long as we can exploit structure such as concavity (or convexity), or linearity.

The problem is that there are many resource allocation problems where the resource vector is not the only information in the state variable. We might have additional information covering a host of activities: temperature, weather forecasts, market prices, the humidity in a laboratory, competitor behavior, ..., the list can be endless. This is information that we would put in our information state variable I_t (see section 9.4).

We need to emphasize that we would only include information in I_t if it is changing over time and, of course, if it affects the behavior of our system. It does not matter if our decisions do or do not affect the trajectory of I_t. When we have an information state variable, then our system state variable is given by (R_t, I_t). The difficulty is that while the value function may be concave (or convex) in R_t, this property typically does not translate to I_t. In particular, we handle I_t as if it is just additional dimensions to R_t, since I_t typically affects how *each* element of R_t affects the value function. So, if we were using our linear value function approximation with slope $\bar{v}_{ta}^n$, we would now want to write $\bar{v}_{ta}^n(I_t)$ to express the dependence on the information state I_t.

Imagine, for example, that we can express our information state as a (not too large) set $\mathcal{I} = \{i_1, i_2, \ldots, i_{|\mathcal{I}|}\}$. Instead of estimating a single value $\bar{v}_{ta}^n$ for each attribute a, we now have to compute $\bar{v}_{ta}^n(i)$ for $i \in \mathcal{I}$. If there are 10 information states, then we just made our problem 10 times harder. However, I_t could be a multidimensional (and possibly continuous) vector.

There are special cases we can handle. The most important arises when I_{t+1} is independent of I_t (or R_t). For example, I_t might be the attributes of a patient, where we are comfortable assuming that the attributes of the patient arriving at $t+1$ has nothing to do with the patient that arrived at time t. In this case, the post-decision state $S_t^x = R_t^x$, which means we forget the information state. This is important because we are typically estimating a value function approximation around the post-decision state.

The property that I_{t+1} does not depend on I_t is referred to as "interstage independence" in the stochastic programming community. While convenient, it does not happen very often. Not surprisingly, this issue arises frequently in machine learning. We saw this earlier in section 7.13.6 for a class of active learning problem (also known as a multiarmed bandit problem) known as the *contextual bandit* problem. This discussion offered a novel perspective of this problem, but otherwise did not offer a solution.

A potential solution approach is to draw a page from our work on hierarchical aggregation. Assume we can create a family of information state variables $(I_t^{(0)}, I_t^{(1)}, \ldots, I_t^{(G)})$ where $I_t^{(0)}$ is our original complete information state, while $I_t^{(g)}$ is a series of successively more aggregate variables for $g = 1, 2, \ldots G$. Assume that each of these variables can be discretized into a set $\mathcal{I}^{(g)}$ of decreasing size. Finally assume that the set $\mathcal{I}^{(G)}$ is relatively small, meaning that we have no difficult creating estimates for each value in $\mathcal{I}^{(G)}$. We simulate I_t and the identify the corresponding elements in each set $\mathcal{I}^{(g)}$. We can then apply our methods of hierarchical aggregation to create a weighted estimate.

18.9 CLOSING NOTES

This chapter has highlighted a variety of complex resource allocation problems, but our examples are all limited to what are known as *single layer* resource allocation problems. For example, we are managing water in reservoirs, money, blood (of different types), and trucks. These problems lend themselves to the kinds of convex approximations described in this chapter.

Imagine now that we want to manage blood, and we have to serve two types of patients: those requiring emergency surgeries that have to be satisfied now, and elective surgeries that can be delayed. In the case of elective surgeries, we have two classes of resources: the blood and the (elective) patients. Without the presence of elective patients, our post-decision state variables would consist only of the different types of blood.

It is *much* harder to approximate the value of the blood resource vector R_t^{blood} when we have the ability to delay elective surgeries, since now the marginal value of an extra unit of blood with attribute a depends on the set of elective surgeries. Separable approximations are unlikely to work, and as interactions become more complex, we need substantially more Benders cuts to capture these. Requiring integer solutions adds additional complexity.

When the future becomes sufficiently complicated, we often have to turn to direct lookahead policies, which we introduce next in chapter 19.

18.10 BIBLIOGRAPHIC NOTES

Section 18.1.1 - In operations research, the newsvendor problem (previously known as the "single period inventory problem" or "the newsboy problem"), arises throughout stochastic resource allocation problems. It is a simple problem that makes it useful for illustrating concepts in stochastic optimization. Yan Qing, Ruoxuan Wang, Asoo Vakharia, Yuwen Chen (2011) provides a good general review; Petruzzi & Dada (1999) provides a somewhat older review (but much of the research was done decades ago). There is still continued interest in specialized topics; for example, DeYong (2020) reviews the newsvendor literature related to price-setting research.

Section 18.1.2 - There is an extensive literature exploiting the natural convexity of $Q(x_0, W_1)$ in x_0, starting with Van Slyke & Wets (1969), followed by the semi-

nal papers on stochastic decomposition (Higle & Sen 1991) and the stochastic dual dynamic programming (SDDP) (Pereira & Pinto 1991). A substantial literature has unfolded around this work, including Shapiro (2011) who provides a careful analysis of SDDP, and its extension to handle risk measures (Shapiro et al. (2013), Philpott et al. (2013)). A number of papers have been written on convergence proofs for Benders-based solution methods, but the best is Girardeau et al. (2014). Kall & Wallace (2009) and Birge & Louveaux (2011) are excellent introductions to the field of stochastic programming. King & Wallace (2012) is a nice presentation on the process of modeling problems as stochastic programs. A modern overview of the field is given by Shapiro et al. (2014).

Section 18.1.3 - The notation in this section was developed in Powell et al. (2001), and applied in a number of papers, including Simao et al. (2009) (for truckload trucking) and Bouzaiene-Ayari et al. (2016) (for a locomotive management problem).

Section 18.2 - The decision of whether to estimate the value function or its derivative is often overlooked in the dynamic programming literature, especially within the operations research community. In the controls community, use of gradients is sometimes referred to as *dual heuristic dynamic programming* (see Werbos (1992) and Venayagamoorthy & Harley (2002)). The operations research community is very familiar with the idea of using marginal values (see, for example, the methods cited in sections 18.5 and 18.6), while the computer science community (among others), works almost exclusively with problems (such as those defined over graphs) where we need the value of being in a state (rather than the marginal value).

Section 18.3 - The CAVE algorithm was first proposed in Godfrey & Powell (2001) for the newsvendor problem, and then extended to spatial resource allocation problems in fleet management in Powell & Godfrey (2002) and Godfrey & Powell (2002). The theory behind the projective SPAR algorithm is given in Powell et al. (2004). A proof of convergence of the leveling algorithm is given in Topaloglu & Powell (2003). A convergence proof for a version of the piecewise linear, separable approximation is given in Zhou et al. (2020).

Section 18.6 - The first paper to formulate a math program with uncertainty appears to be Dantzig & Ferguson (1956). For a broad introduction to the field of stochastic optimization, see Ermoliev (1988) and Pflug (1996). For complete treatments of the field of stochastic programming, see Shapiro (2003), Birge & Louveaux (2011), Kall & Mayer (2005), and Shapiro et al. (2014). For an easy tutorial on the subject, see Sen & Higle (1999). A very thorough introduction to stochastic programming is given in Ruszczyński & Shapiro (2003). Mayer (1998) provides a detailed presentation of computational work for stochastic programming. There has been special interest in the types of network problems we have considered (see Wallace (1986), S.W. & Wallace (1987), and Birge et al. (1988)). Rockafellar & Wets (1991) presents specialized algorithms for stochastic programs formulated using scenarios. This modeling framework has been of particular interest in the are of financial portfolios (Mulvey & Ruszczyński (1995)). Benders' decomposition for two-stage stochastic programs was first proposed by Van Slyke & Wets (1969) as the "L-shaped" method. Higle & Sen (1991) introduce stochastic decomposition, which is a Monte-Carlo based algorithm that is most similar in spirit to approximate dynamic programming. Chen & Powell (1999) present a variation of Benders that falls between stochastic

decomposition and the L-shaped method. The relationship between Benders' decomposition and dynamic programming is often overlooked. A notable exception is Pereira & Pinto (1991), which uses Benders to solve a resource allocation problem arising in the management of reservoirs. This paper presents Benders as a method for avoiding the curse of dimensionality of dynamic programming. For an excellent review of Benders' decomposition for multistage problems, see Ruszczyński (2003). Benders has been extended to multistage problems in Birge (1985), Ruszczyński (1993), and Chen & Powell (1999), which can be viewed as a form of approximate dynamic programming using cuts for value function approximations.

Section 18.7 - High-dimensional applications arise when, for example, we need to estimate $\bar{v}_a$ where $a \in \mathcal{A}$ is a multidimensional vector where the set $\mathcal{A}$ can have a number of elements that far exceeds our budget for observations. This section used hierarchical learning which was developed in George et al. (2008) VFA paper (and described in section 3.6. These methods were used in Simao et al. (2009) where a is the attributes of a truck driver.

Section 18.8 - The stochastic optimization literature has long realized that it is relatively easy to approximate a function $V(R)$ which is concave (convex if minimizing) in R, and where R may be high dimensional. However, there are many problems that involve managing resource allocation problems that combine a resource vector R_t with an exogenously evolving information state I_t, which means the state of the system is $S_t = (R_t, I_t)$. I_t is often relatively unstructured data such as weather, prices, forecasts, the humidity in a laboratory, and so on. The stochastic programming community often assumes "interstage independence" which means that the post-decision state $S_t^x = R_t^x$ (that is, it does not depend on I_t); see Morton (1996), Queiroza & Morton (2013). Asamov & Powell (2018) presents a regularization algorithm that assumes that I_t can take on a "finite" (that is, not too large) number of discrete values $I_1, I_2, \ldots, I_K$.

EXERCISES

Review questions

18.1 Give three examples of resource allocation problems not listed in table 18.1. Describe the type of resource (or resources) and the decisions that need to be made.

18.2 What is meant by a "two-stage" resource allocation problem? Given an example.

18.3 Equations (18.12) - (18.15) use variables such as $x_{t+1}(\omega)$, $A_{t+1}(\omega)$ and $b_{t+1}(\omega)$? What is meant by ω, and what do we mean when we write it as an argument as we did with these three variables?

18.4 Imagine that we have to solve a newsvendor problem with dynamically varying costs and prices:

$$\max_{x \leq R_t} F(x) = \mathbb{E}\left\{p_t \max\{x, W_{t+1}\} - c_t x | S_t\right\}$$

What is the resource state variable? What are the "exogenous information state" variables?

Modeling questions

18.5 Following the general modeling style of section 18.1.3, create your own model of a fleet of autonomous electric vehicles, where the goal is to simulate the dispatching process over the course of the day. The following are some general guidelines to follow when creating your model:

- Assume that you are modeling a region (say a state) that has been divided into a set of zones $z \in \mathcal{Z}$. There may be anywhere between 100 and 10,000 zones depending on the size of the region and the size of the zones.
- Assume you are modeling time in 15 minute increments, over an entire day.
- You will need to model a fleet of vehicles $i \in \mathcal{I}$.
- Let b_{ti} be the battery charge level in vehicle i at time t. You may assume that all vehicles are fully charged at the beginning of the day. Let η^{move} be the rate that each car consumes energy while moving, and η^{idle} for the energy consummption rate while sitting idle.
- Let a_{ti} be the characteristics of vehicle i at time t, which will include current location (if idle), or the location it is heading to if in the middle of a trip; the time period it is expected to arrive (if moving); and the battery charge level b_{ti}.
- Let $\hat{D}_{t+1,zz'}$ be the number of new requests for trips that arrive between t and $t+1$ to travel from zone z to zone z'. Let $x_{tdi} = 1$ if we choose to implement decision $d \in \mathcal{D}$ for vehicle i.
- You will need to introduce a set of decisions $\mathcal{D}$ where $d \in \mathcal{D}$ can be: move to pick up a customer, move empty, do nothing, move to a recharging station to be recharged (introduce notation for recharging stations).

Set up all five elements of a dynamic model, using $X^\pi(S_t)$ as your policy. Then, suggest two policies that you think might work, one from the policy search class, and one from the lookahead class.

Computational exercises

18.6 Consider a newsvendor problem where we solve

$$\max_x \mathbb{E}F(x, \hat{D}),$$

where

$$F(x, \hat{D}) = p \min(x, \hat{D}) - cx.$$

We have to choose a quantity x before observing a random demand $\hat{D}$. For our problem, assume that $c = 1$, $p = 2$, and that $\hat{D}$ follows a discrete uniform distribution between 1 and 10 (that is, $\hat{D} = d, d = 1, 2, \ldots, 10$ with probability 0.10). Approximate $\mathbb{E}F(x, \hat{D})$ as a piecewise linear function using the methods described in section 18.3, using a stepsize $\alpha_{n-1} = 1/n$. Note that you are using derivatives of $F(x, \hat{D})$ to estimate the slopes of the

function. At each iteration, randomly choose x between 1 and 10. Use sample realizations of the gradient to estimate your function. Compute the exact function and compare your approximation to the exact function.

18.7 Repeat exercise 18.6, but this time approximate $\mathbb{E}F(x, \hat{D})$ using a linear approximation:

$$\bar{F}(x) = \theta x.$$

Compare the solution you obtain with a linear approximation to what you obtained using a piecewise-linear approximation. Now repeat the exercise using demands that are uniformly distributed between 500 and 1000. Compare the behavior of a linear approximation for the two different problems.

18.8 Repeat exercise 18.6, but this time approximate $\mathbb{E}F(x, \hat{D})$ using the SHAPE algorithm. Start with an initial approximation given by

$$\bar{F}^0(x) = \theta_0(x - \theta_1)^2.$$

Use the recursive regression methods of sections 18.4 and 3.8 to fit the parameters. Justify your choice of stepsize rule. Compute the exact function and compare your approximation to the exact function.

18.9 Repeat exercise 18.6, but this time approximate $\mathbb{E}F(x, \hat{D})$ using the regression function given by

$$\bar{F}(x) = \theta_0 + \theta_1 x + \theta_2 x^2.$$

Use the recursive regression methods of sections 18.4 and 3.8 to fit the parameters. Justify your choice of stepsize rule. Compute the exact function and compare your approximation to the exact function. Estimate your value function approximation using two methods:

(a) Use observations of $F(x, \hat{D})$ to update your regression function.

(b) Use observations of the derivative of $F(x, \hat{D})$, so that $\bar{F}(x)$ becomes an approximation of the derivative of $\mathbb{E}F(x, \hat{D})$.

18.10 Approximate the function $\mathbb{E}F(x, \hat{D})$ in exercise 18.6, but now assume that the random variable $\hat{D} = 1$ (that is, it is deterministic). Using the following approximation strategies:

(a) Use a piecewise linear value function approximation. Try using both left and right derivatives to update your function.

(b) Use the regression $\bar{F}(x) = \theta_0 + \theta_1 x + \theta_2 x^2$.

18.11 We are going to solve the basic asset acquisition problem (section 8.2.1) where we purchase assets (at a price p^p) at time t to be used in time interval $t + 1$. We sell assets at a price p^s to satisfy the demand $\hat{D}_t$ that arises during time interval t. The problem is to be solved over a finite time horizon T. Assume that the initial inventory is 0 and that demands follow a discrete uniform distribution over the range $[0, D^{max}]$. The problem parameters

are given by

$$\begin{aligned}\gamma &= 0.8,\\ D^{max} &= 10,\\ T &= 20,\\ p^p &= 5,\\ p^s &= 8.\end{aligned}$$

Solve this problem by estimating a piecewise linear value function approximation (section 18.3). Choose $\alpha_{n+1} = a/(a+n)$ as your stepsize rule, and experiment with different values of a (such as 1, 5, 10, and 20). Use a single-pass algorithm, and report your profits (summed over all time periods) after each iteration. Compare your performance for different stepsize rules. Run 1000 iterations and try to determine how many iterations are needed to produce a good solution (the answer may be substantially less than 1000).

18.12 Repeat exercise 18.11, but this time use the SHAPE algorithm to approximate the value function. Use as your initial value function approximation the function

$$\overline{V}_t^0(R_t) = \theta_0(R_t - \theta_2)^2.$$

For each of the exercises below, you may have to tweak your stepsize rule. Try to find a rule that works well for you (we suggest sticking with a basic $a/(a+n)$ strategy). Determine an appropriate number of training iterations, and then evaluate your performance by averaging results over 100 iterations (testing iterations) where the value function is not changed.

(a) Solve the problem using $\theta_0 = 1, \theta_1 = 5$.

(b) Solve the problem using $\theta_0 = 1, \theta_1 = 50$.

(c) Solve the problem using $\theta_0 = 0.1, \theta_1 = 5$.

(d) Solve the problem using $\theta_0 = 10, \theta_1 = 5$.

(e) Summarize the behavior of the algorithm with these different parameters.

18.13 Repeat exercise 18.11, but this time assume that your value function approximation is given by

$$\overline{V}_t^0(R_t) = \theta_0 + \theta_1 R_t + \theta_2 R_t^2.$$

Use the recursive regression techniques of sections 18.4 and 3.8 to determine the values for the parameter vector θ.

18.14 Repeat exercise 18.11, but this time assume you are solving an infinite horizon problem (which means you only have one value function approximation).

18.15 Repeat exercise 18.13, but this time assume an infinite horizon.

18.16 Repeat exercise 18.11, but now assume the following problem parameters:

$$\begin{aligned}\gamma &= 0.99,\\ T &= 200,\\ p^p &= 5,\\ p^s &= 20.\end{aligned}$$

For the demand distribution, assume that $\hat{D}_t = 0$ with probability 0.95, and that $\hat{D}_t = 1$ with probability 0.05. This is an example of a problem with low demands, where we have to hold inventory for a fairly long time.

Sequential decision analytics and modeling

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

18.17 Read sections 13.1-13.4 on the blood management problem. An approximate dynamic programming algorithm has been implemented in Python, which can be downloaded from `http://tinyurl.com/sdagithub` using the module "BloodManagement."

a) Use the ADP algorithm to create piecewise linear value function approximations, and simulate the resulting policy. Report the objective function you obtain from simulating the policy.

b) Now set all the VFA's equal to zero and simulate the resulting myopic policy. Compare the performance of the myopic policy to the VFA-based policy.

c) Increase the supply of blood by 50 percent (multiply all input supplies by 1.5), and repeat the comparison of parts a and b. How did the relative value of the VFA-based policy change in the presence of inflated supplies of blood?

d) Repeat part (c), but this time multiply the supplies by 0.80. How does this affect the results?

Diary problem

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

18.18 This chapter is designed for problems that enjoy the property of concavity (convexity if minimizing). This often arises in the context of resource allocation problems, but may arise in other settings as well. Identify any state variables where you believe the value function would be concave (or convex) with respect to this (these) state variables, and if these are present in your problem, describe an approximation architecture that exploits this property.

Bibliography

Asamov, T. & Powell, W. B. (2018), 'Regularized Decomposition of High { Dimensional Multistage Stochastic Programs with Markov Uncertainty', *SIAM J. Optimization* **28**(1), 575–595.

Birge, J. R. (1985), 'Decomposition and partitioning techniques for multistage stochastic linear programs', *Mathematical Programming* **33**, 25–41.

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

Birge, J. R., Wallace, S. W. & S.W.Wallace (1988), 'A Separable Piecewise Linear Upper Bound for Stochastic Linear Programs', *SIAM Journal of Control and Optimization* **26**, 1–14.

Bouzaiene-Ayari, B., Cheng, C., Das, S., Fiorillo, R. & Powell, W. B. (2016), 'From single commodity to multiattribute models for locomotive optimization: A comparison of optimal integer programming and approximate dynamic programming', *Transportation Science* **50**(2), 1–24.

Chen, Z.-L. L. & Powell, W. B. (1999), 'A Convergent Cutting-Plane and Partial-Sampling Algorithm for Multistage Linear Programs with Recourse', *Journal of Optimization Theory and Applications* **103**(3), 497–524.

Dantzig, G. B. & Ferguson, A. (1956), 'The Allocation of Aircrafts to Routes: An Example of Linear Programming Under Uncertain Demand', *Management Science* **3**, 45–73.

DeYong, G. (2020), 'The price-setting newsvendor: review and extensions', *International Journal of Production Research* **58**(6), 1776–1804.

Ermoliev, Y. (1988), Stochastic Quasigradient Methods, *in* Y. Ermoliev & R. Wets, eds, 'Numerical Techniques for Stochastic Optimization', Springer-Verlag, Berlin.

George, A., Powell, W. B. & Kulkarni, S. (2008), 'Value Function Approximation using Multiple Aggregation for Multiattribute Resource Management', *J. Machine Learning Research* pp. 2079–2111.

Girardeau, P., Leclere, V. & Philpott, A. B. (2014), 'On the Convergence of Decomposition Methods for Multistage Stochastic Convex Programs', *Mathematics of Operations Research* **40**(1), 130–145.

Godfrey, G. A. & Powell, W. B. (2001), 'An Adaptive, Distribution-Free Algorithm for the Newsvendor Problem with Censored Demands, with Applications to Inventory and Distribution', *Management Science* **47**(8), 1101–1112.

Godfrey, G. A. & Powell, W. B. (2002), 'An Adaptive Dynamic Programming Algorithm for Dynamic Fleet Management, II: Multiperiod Travel Times', *Transportation Science* **36**(1), 40–54.

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

Kall, P. & Mayer, J. (2005), *Stochastic Linear Programming: Models, Theory, and Computation*, Springer, New York.

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

King, A. J. & Wallace, S. W. (2012), *Modeling with Stochastic Programming*, Springer Verlag, New York.

Mayer, J. (1998), *Stochastic linear programming algorithms: A comparison based on a model management system*, Springer.

Morton, D. P. (1996), 'An enhanced decomposition algorithm for multistage stochastic hydroelectric scheduling', *Annals of Operations Research* **64**, 211–235.

Mulvey, J. M. & Ruszczyński, A. (1995), 'A new scenario decomposition method for large scale stochastic optimization', *Operations Research* **43**, 477–490.

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

Petruzzi, N. C. & Dada, M. (1999), 'Pricing and the Newsvendor Problem: A Review with Extensions', *Operations Research* **47**, 183–194.

Pflug, G. (1996), *Optimization of Stochastic Models: The Interface Between Simulation and Optimization*, Kluwer International Series in Engineering and Computer Science: Discrete Event Dynamic Systems, Kluwer Academic Publishers, Boston.

Philpott, A. B., De Matos, V. & Finardi, E. (2013), 'On Solving Multistage Stochastic Programs with Coherent Risk Measures', *Operations Research* **51**(4), 957–970.

Powell, W. B. & Godfrey, G. A. (2002), 'An adaptive dynamic programming algorithm for dynamic fleet management, I: Single period travel times', *Transportation Science* **36**(1), 40–54.

Powell, W. B., Ruszczyński, A. & Topaloglu, H. (2004), 'Learning Algorithms for Separable Approximations of Discrete Stochastic Optimization Problems', *Mathematics of Operations Research* **29**(4), 814–836.

Powell, W. B., Simao, H. P. & Shapiro, J. A. (2001), A representational paradigm for dynamic resource transformation problems, *in* F. C. Coullard & J. Owens, H., eds, 'Annals of Operations Research', J. C. Baltzer AG, pp. 231–279.

Queiroza, A. R. & Morton, D. P. (2013), 'Sharing cuts under aggregated forecasts when decomposing multi-stage stochastic programs', *Operations Research Letters* **41**(3), 311–316.

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

Ruszczyński, A. (1993), 'Parallel Decomposition of Multistage Stochastic Programming Problems', *Mathematical Programming* **58**, 201–228.

Ruszczyński, A. (2003), *Decomposition Methods*, Elsevier, Amsterdam.

Ruszczyński, A. & Shapiro, A. (2003), *Handbooks in Operations Research and Management Science: Stochastic Programming*, Vol. 10, Elsevier, Amsterdam.

Sen, S. & Higle, J. L. (1999), 'An Introductory Tutorial on Stochastic Linear Programming Models', *Interfaces* (April), 33–61.

Shapiro, A. (2003), *Stochastic Programming*, Vol. 10, John Wiley & Sons, Chichester, U.K.

Shapiro, A. (2011), 'Analysis of stochastic dual dynamic programming method', *European Journal of Operational Research* **209**(1), 63–72.

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

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

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

S.W., W. & Wallace, S. W. (1987), 'A piecewise linear upper bound on the network recourse function', *Mathematical Programming* **38**, 133–146.

Topaloglu, H. & Powell, W. B. (2003), 'An Algorithm for Approximating Piecewise Linear Concave Functions from Sample Gradients', *Operations Research Letters* **31**, 66–76.

Van Slyke, R. M. & Wets, R. J.-B. (1969), 'L-shaped linear programs with applications to optimal control and stochastic programming', *SIAM Journal of Applied Mathematics* **17**, 638–663.

Venayagamoorthy, G. & Harley, R. (2002), 'Comparison of heuristic dynamic programming and dual heuristic programming adaptive critics for neurocontrol of a turbogenerator', *Neural Networks, IEEE* **13**(3), 764–773.

Wallace, S. W. (1986), 'Solving stochastic programs with network recourse', *Networks* **16**, 295–317.

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

Yan Qing, Ruoxuan Wang, Asoo Vakharia, Yuwen Chen, M. S. (2011), 'The newsvendor problem: Review and directions for future research', *European J. Operation Research* **213**, 361–374.

Zhou, S., Wang, F., Shi, N. & Xu, Z. (2020), 'A New Convergent Hybrid Learning Algorithm for Two-Stage Stochastic Programs', *European Journal of Operational Research* **283**(1), 33–46.