TUTORIALS IN
OPERATIONS RESEARCH
INFORMS—New Orleans 2005

 | ISBN 0000-0000
DOI 10.1287/educ.1053.0000

Chapter X

Approximate Dynamic Programming for Large-Scale Resource Allocation Problems

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

Huseyin Topaloglu
School of Operations Research and Industrial Engineering,
Cornell University, Ithaca, New York 14853, USA,
topaloglu@orie.cornell.edu

Abstract We present modeling and solution strategies for large-scale resource allocation problems that take place over multiple time periods under uncertainty. In general, the strategies we present formulate the problem as a dynamic program and replace the value functions with tractable approximations. The approximations of the value functions are obtained by using simulated trajectories of the system and iteratively improving on (possibly naive) initial approximations; we propose several improvement algorithms for this purpose. As a result, the resource allocation problem decomposes into time-staged subproblems, where the impact of the current decisions on the future evolution of the system is assessed through value function approximations. Computational experiments indicate that the strategies we present yield high-quality solutions. We also present comparisons with conventional stochastic programming methods.

Keywords dynamic programming; approximate dynamic programming; stochastic approximation; large-scale optimization

1. Introduction

Many problems in operations research can be posed as managing a set of resources over multiple time periods under uncertainty. The resources may take on different forms in different applications; vehicles and containers for fleet management, doctors and nurses for personnel scheduling, cash and stocks for financial planning. Similarly, the uncertainty may have different characterizations in different applications; load arrivals and weather conditions for fleet management, patient arrivals for personnel scheduling, interest rates for financial planning. Despite the differences in terminology and application domain, a unifying aspect of these problems is that we have to make decisions under the premise that the decisions that we make now will affect the future evolution of the system and the future evolution of the system is also affected by random factors that are beyond our control.

A classical approach for solving such problems is to use the theory of Markov decision processes. The fundamental idea is to use a state variable that represents all information that is relevant to the future evolution of the system. Given the current value of the state variable, value functions capture the total expected cost incurred by the system over the whole planning horizon. Unfortunately, time and storage requirements for computing the value functions through conventional approaches, such as value iteration and policy iteration, increase exponentially with the number of dimensions of the state variable. For the applications above, these conventional approaches are simply intractable.

This chapter presents a modeling framework for large-scale resource allocation problems, along with a fairly flexible algorithmic framework that can be used to obtain good solutions for them. Our modeling framework is motivated by transportation applications, but it provides enough generality to capture a variety of other problem settings. We do not focus on a specific application domain throughout the chapter, although we use the transportation setting to give concrete examples. The idea behind our algorithmic framework is to formulate the problem as a dynamic program and to use tractable approximations of the value functions, which are obtained by using simulated trajectories of the system and iteratively improving on (possibly naive) initial value function approximations.

The organization of the chapter is as follows. Sections 2 and 3 respectively present our modeling and algorithmic frameworks for describing and solving resource allocation problems. Section 4 describes a variety of methods that one can use to improve on the initial value function approximations. Section 5 focuses on the stepsize choices for the methods described in Section 4. In Section 6, we review other possible approaches for solving resource allocation problems, most of which are motivated by the field of stochastic programming. Section 7 presents some computational experiments. We conclude in Section 8 with possible extensions and unresolved issues.

2. Modeling Framework

This section describes a modeling framework for resource allocation problems. Our approach borrows ideas from mathematical programming, probability theory and computer science. This modeling framework has been beneficial to us for several reasons. First, it offers a modeling language that is independent of the problem domain; one can use essentially the same language to describe a problem that involves assigning trucks to loads or a problem that involves scheduling computing tasks on multiple servers. Second, it extensively uses terminology, such as resources, decisions, transformation and information, that is familiar to nonspecialists. This enables us to use our modeling framework as a communication tool when talking to a variety of people. Third, it is software-friendly; the components of our modeling framework can easily be mapped to software objects. This opens the door for developing general purpose software that can handle a variety of resource allocation problems.

We present our modeling framework by summarizing the major elements of a Markov decision process, ending with a formal statement of our objective function. However, working with this objective function is computationally intractable and we focus on an approximation strategy in Section 3.

2.1. Modeling Time

Perhaps one of the most subtle dimensions of modeling a stochastic optimization problem is the modeling of time. In a stochastic model of a resource allocation problem, there are two processes taking place; the flow of physical resources and the flow of information. The flow of information can be further divided between the flow of exogenous information and the flow of decisions.

For computational reasons, we assume that decisions are made at discrete points in time. These points in time, known as decision epochs, might be once every week, once every four hours or once every second. They may also be determined by exogenous events, such as phone calls or arrivals of customers, in which case the time interval between the decision epochs is not constant.

In contrast, the arrival of exogenous information and the movement of resources occurs in continuous time. We might, for example, approximate a transportation problem by assuming that the decisions are made once every four hours, but the actual movements of the physical resources still occurs in continuous time between the decision epochs. It is notationally convenient to represent the decision epochs with the integers $\mathcal{T} = \{0, 1, \ldots, T\}$ where T is

the end of our planning horizon. Physical activities, such as arrivals of customers, departures of aircraft, job completions, and the arrival of information, such as customer requests, equipment failures, notifications of delays, can occur at continuous points in time between these decision epochs.

2.2. Resources

We use a fairly general notation to model resources which handles both simple resources, such as oil, money, agricultural commodities, and complex resources, such as people, specialized machinery. We represent resources using

$\mathcal{A}$ = Attribute space of the resources. We usually use a to denote a generic element of the attribute space and refer to $a = (a_1, a_2, \ldots, a_I)$ as an attribute vector.
R_{ta} = Number of resources with attribute vector a at time period t just before a decision is made.
$R_t = (R_{ta})_{a \in \mathcal{A}}$.

Roughly speaking, the attribute space represents the set of all possible states that a particular resource can be in. For example, letting $\mathcal{I}$ be the set of locations in the transportation network and $\mathcal{V}$ be the set of vehicle types, and assuming that the maximum travel time between any origin-destination pair is τ time periods, the attribute space of the vehicles in the fleet management setting is $\mathcal{A} = \mathcal{I} \times \{0, 1, \ldots, \tau\} \times \mathcal{V}$. A vehicle with the attribute vector

$$a = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} \text{inbound/current location} \\ \text{time to reach inbound location} \\ \text{vehicle type} \end{bmatrix} \tag{1}$$

is a vehicle of type a_3 that is inbound to (or at) location a_1 and that will reach location a_1 at time a_2 (it is in the attribute a_2 that we model time continuously). The attribute a_2 might also be the time remaining until the vehicle is expected to arrive, or it might even be the departure time from the origin (this might be needed if the travel time is random). We note that certain attributes can be dynamic, such as inbound/current location, and certain attributes can be static, such as vehicle type. We access the number of vehicles with attribute vector a at time period t by referring to R_{ta}. This implies that we can "put" the vehicles with the same attribute vector in the same "bucket" and treat them as indistinguishable.

We assume that our resources are being used to serve demands; for example demands for finishing a job, moving a passenger or carrying a load of freight. We model the demands using

$\mathcal{B}$ = Attribute space of the demands. We usually use b to denote a generic element of the attribute space.
D_{tb} = Number of demands with attribute vector b waiting to be served at time period t.
$D_t = (D_{tb})_{b \in \mathcal{B}}$.

To keep the notation simple, we assume that any unserved demands are immediately lost.

Although we mostly consider the case where the resources are indivisible and R_{ta} takes integer values, R_{ta} may in fact be allowed to take fractional values. For example, R_{ta} may represent the inventory level of a certain type of product at time period t measured in kilograms. Also, we mostly consider the case where the attribute space is finite. Finally, the definition of the attribute space implies that the resources we are managing are uniform; that is, the attribute vector for each resource takes values in the same space. However,

by defining multiple attribute spaces, say $\mathcal{A}^1, \ldots, \mathcal{A}^N$, we can deal with multiple types of resources. For example, $\mathcal{A}^1$ may correspond to the drivers, whereas $\mathcal{A}^2$ may correspond to the trucks.

The attribute vector is a flexible object that allows us to model a variety of situations. In the fleet management setting with single-period travel times and a homogenous fleet, the attribute space is as simple as $\mathcal{I}$. On the other extreme, we may be dealing with vehicles with the attribute vector

$$\begin{bmatrix} \text{inbound/current location} \\ \text{time to reach inbound location} \\ \text{duty time within shift} \\ \text{days away from home} \\ \text{vehicle type} \\ \text{home domicile} \end{bmatrix}. \tag{2}$$

Based on the nature of the attribute space, we can model a variety of well-known problem classes.

1. Single-product inventory control problems. If the attribute space is a singleton, say $\{a\}$, then R_{ta} simply gives the inventory count at time period t.

2. Multiple-product inventory control problems. If we have $\mathcal{A} = \{1, \ldots, N\}$ and the attributes of the resources are static (product type), then R_{ta} gives the inventory count for product type a at time period t.

3. Single-commodity min-cost network flow problems. If we have $\mathcal{A} = \{1, \ldots, N\}$ and the attributes of the resources are dynamic, then R_{ta} gives the number of resources in state a at time period t. For example, this type of a situation arises when one manages a homogenous fleet of vehicles whose only attributes of interest are their locations. Our terminology is motivated by the fact that the deterministic versions of these problems can be formulated as min-cost network flow problems.

4. Multicommodity min-cost network flow problems. If we have $\mathcal{A} = \{1, \ldots, I\} \times \{1, \ldots, K\}$, and the first element of the attribute vector is static and the second element is dynamic, then $R_{t,[i,k]}$ gives the number of resources of type i that are in state k at time period t. For example, this type of a situation arises when one manages a heterogeneous fleet of vehicles whose only attributes of interest are their sizes (i) and locations (k).

5. Heterogeneous resource allocation problems. This is a generalization of the previous problem class where the attribute space involves more than two dimensions, some of which are static and some of which are dynamic.

From a purely mathematical viewpoint, since we can "lump" all information about a resource into one dynamic attribute, single-commodity min-cost network flow problems provide enough generality to capture the other four problem classes. However, from the algorithmic viewpoint, the solution methodology we use and our ability to obtain integer solutions depend very much on what problem class we work on. For example, we can easily enumerate all possible attribute vectors in $\mathcal{A}$ for the first four problem classes, but this may not be possible for the last problem class. When obtaining integer solutions is an issue, we often exploit a network flow structure. This may be possible for the first three problem classes, but not for the last two.

We emphasize that the attribute space is different than what is commonly referred to as the state space in Markov decision processes. The attribute space represents the set of all possible states that a particular resource can be in. On the other hand, the state space in Markov decision processes refers to the set of all possible values that the resource state vector R_t can take. For example, in the fleet management setting, the number of elements of the attribute space $\mathcal{A} = \mathcal{I} \times \{0, 1, \ldots \tau\} \times \mathcal{V}$ is on the order of several thousands. On the other hand, the state space includes all possible allocations of the fleet among different locations, which is an intractable number even for problems with a small number of vehicles in the fleet, a small number of locations and a small number of vehicle types.

2.3. Evolution of Information

We define

$$\begin{aligned}\hat{R}_{ta}(R_t) &= \text{Random variable representing the change in the number of resources with attribute vector } a \text{ that occurs during time period } t.\\ \hat{R}_t(R_t) &= (\hat{R}_{ta}(R_t))_{a\in\mathcal{A}}.\end{aligned}$$

The random changes in the resource state vector may occur due to new resource arrivals or changes in the status of the existing resources. For notational brevity, we usually suppress the dependence on R_t. We model the flow of demands in a similar way by defining

$$\begin{aligned}\hat{D}_{tb}(R_t) &= \text{Random variable representing the new demands with attribute vector } b \text{ that become available during time period } t.\\ \hat{D}_t(R_t) &= (\hat{D}_{tb}(R_t))_{b\in\mathcal{B}}.\end{aligned}$$

From time to time, we need a generic variable to represent all the exogenous information that become available during time period t. The research community has not adopted a standard notation for exogenous information; we use

$$W_t = \text{Exogenous information that become available during time period } t.$$

For our problem, we have $W_t = (\hat{R}_t, \hat{D}_t)$.

2.4. The State Vector

The state vector captures the information that we need at a certain time period to model the future evolution of the system. Virtually every textbook on dynamic programming represents the state vector as the information available just before we make the decisions. If we let S_t be the state of the system just before we make the decisions at time period t, then we have

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

We refer to S_t as the pre-decision state vector to emphasize that it is the state of the system just before we make the decisions at time period t. To simplify our presentation, we assume that any unserved demands are lost, which means that $D_t = \hat{D}_t$. We will also find it useful to use the state of the system immediately after we make the decisions. We let

$$R_t^x = \text{The resource state vector immediately after we make the decisions at time period } t.$$

Since we assume that any unserved demands are lost, the state of the system immediately after we make the decisions at time period t is given by

$$S_t^x = R_t^x.$$

We refer to S_t^x as the post-decision state vector. For notational clarity, we often use R_t^x to capture the post-decision state vector.

It helps to summarize the sequence of states, decisions and information by using

$$(S_0, x_0, S_0^x, W_1, S_1, x_1, S_1^x, \ldots, W_t, S_t, x_t, S_t^x, \ldots, W_T, S_T, x_T, S_T^x),$$

where x_t is the decision vector at time period t.

2.5. Decisions

Decisions are the means by which we can modify the attributes of the resources. We represent the decisions by defining

$\mathcal{C}$ = Set of decision classes. We can capture a broad range of resource allocation problems by using two classes of decisions; D to serve a demand and M to modify a resource without serving a demand.
$\mathcal{D}^D$ = Set of decisions to serve a demand. Each element of $\mathcal{D}^D$ represents a decision to serve a demand with a particular attribute vector; that is, there is an attribute vector $b_d \in \mathcal{B}$ for each $d \in \mathcal{D}^D$.
$\mathcal{D}^M$ = Set of decisions to modify a resource without serving a demand. In the transportation setting, this often refers to moving a vehicle from one location to another, but it can also refer to repairing the vehicle or changing its configuration. We assume that one element of $\mathcal{D}^M$ is a decision that represents "doing nothing."
$\mathcal{D} = \mathcal{D}^D \cup \mathcal{D}^M$.
x_{tad} = Number of resources with attribute vector a that are modified by using decision d at time period t.
c_{tad} = Profit contribution from modifying one resource with attribute vector a by using decision d at time period t.

Using standard terminology, $x_t = (x_{tad})_{a\in\mathcal{A},d\in\mathcal{D}}$ is the decision vector at time period t, along with the objective coefficients $c_t = (c_{tad})_{a\in\mathcal{A},d\in\mathcal{D}}$. If it is infeasible to apply decision d on a resource with attribute vector a, then we capture this by letting $c_{tad} = -\infty$. Fractional values may be allowed for x_{tad}, but we mostly consider the case where x_{tad} takes integer values.

In this case, the resource conservation constraints can be written as

$$\sum_{d\in\mathcal{D}} x_{tad} = R_{ta} \qquad \text{for all } a \in \mathcal{A}. \tag{3}$$

These constraints simply state that the total number of resources with attribute vector a that are modified by using a decision at time period t equals the number of resources with attribute vector a.

Typically there is a reward for serving a demand, but the number of such decisions is restricted by the number of demands. Noting that $d \in \mathcal{D}^D$ represents a decision to serve a demand with attribute vector b_d, we write the demand availability constraints as

$$\sum_{a\in\mathcal{A}} x_{tad} \le \hat{D}_{t,b_d} \qquad \text{for all } d \in \mathcal{D}^D.$$

We can now write our set of feasible decisions as

$$\mathcal{X}(S_t) = \Big\{ x_t : \sum_{d\in\mathcal{D}} x_{tad} = R_{ta} \quad \text{for all } a \in \mathcal{A} \tag{4}$$

$$\sum_{a\in\mathcal{A}} x_{tad} \le \hat{D}_{t,b_d} \quad \text{for all } d \in \mathcal{D}^D \tag{5}$$

$$x_{tad} \in \mathbb{Z}_+ \qquad \text{for all } a \in \mathcal{A},\ d \in \mathcal{D} \Big\}. \tag{6}$$

Our challenge is to find a policy or decision function that determines what decisions we should take. We let

$X_t^\pi(\cdot)$ = A function that maps the state vector S_t to the decision vector x_t at time period t; that is, we have $X_t^\pi(S_t) \in \mathcal{X}(S_t)$.

There can be many choices for this function; we dwell on this issue in Section 3.

2.6. Transition Function

We capture the result of applying decision d on a resource with attribute vector a by

$$\delta_{a'}(a,d) = \begin{cases} 1 & \text{If applying decision } d \text{ on a resource with attribute vector } a \text{ transforms the resource into a resource with attribute vector } a' \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

Using the definition above, the resource dynamics can be written as

$$\begin{aligned} R^x_{ta} &= \sum_{a' \in \mathcal{A}} \sum_{d \in \mathcal{D}} \delta_a(a',d)\, x_{ta'd} && \text{for all } a \in \mathcal{A} \\ R_{t+1,a} &= R^x_{ta} + \hat{R}_{t+1,a} && \text{for all } a \in \mathcal{A}. \end{aligned} \tag{8}$$

It is often useful to represent the system dynamics generically using

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

where $W_{t+1} = (\hat{R}_{t+1}, \hat{D}_{t+1})$ is the new information arriving during time period $t+1$. Therefore, $S^M(\cdot,\cdot)$ is a function that maps the decision vector and the new information to a state vector for the next time period.

2.7. Objective Function

We are interested in finding decision functions $\{X^\pi_t(\cdot) : t \in \mathcal{T}\}$ that maximize the total expected profit contribution over the planning horizon. Noting that a set of decision functions $\{X^\pi_t(\cdot) : t \in \mathcal{T}\}$ define a policy π and letting Π be the set of all possible policies, we want to solve

$$\max_{\pi \in \Pi} \mathbb{E}\left\{ \sum_{t \in \mathcal{T}} C_t(X^\pi_t(S_t)) \right\}, \tag{9}$$

where we let $C_t(x_t) = \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} c_{tad}\, x_{tad}$ for notational brevity. The problem above is virtually impossible to solve directly. The remainder of this chapter focuses on describing how approximate dynamic programming can be used to find high-quality solutions to this problem.

3. An Algorithmic Framework for Approximate Dynamic Programming

It is well-known that an optimal policy that solves problem (9) satisfies the Bellman equation

$$V_t(S_t) = \max_{x_t \in \mathcal{X}(S_t)} C_t(x_t) + \mathbb{E}\left\{ V_{t+1}(S^M(S_t, x_t, W_{t+1})) \,|\, S_t \right\}. \tag{10}$$

It is also well-known that solving problem (10) suffers from the so-called curse of dimensionality. It is typically assumed that we have to solve (10) for every possible value of the state vector S_t. When S_t is a high-dimensional vector, the number of possible values for S_t quickly becomes intractably large. For our problems, S_t may have hundreds of thousands of dimensions.

Unfortunately, the picture is worse than it seems at the first sight; there are actually three curses of dimensionality. The first one is the size of the state space, which explodes when S_t is a high-dimensional vector. The second one is the size of the outcome space that becomes problematic when we try to compute the expectation in (10). This expectation is often hidden in the standard textbook representations of the Bellman equation, which is written as

$$V_t(S_t) = \max_{x_t \in \mathcal{X}(S_t)} C_t(x_t) + \sum_{s' \in \mathcal{S}} p(s' \,|\, S_t, x_t)\, V_{t+1}(s'),$$

where $\mathcal{S}$ is the set of all possible values for the state vector S_{t+1} and $p(s' \mid S_t, x_t)$ is the probability that $S^M(S_t, x_t, W_{t+1}) = s'$ conditional on S_t and x_t. Most textbooks on dynamic programming assume that the transition probability $p(s' \mid S_t, x_t)$ is given, but in many problems such as ours, it can be extremely difficult to compute.

The third curse of dimensionality is the size of the action space $\mathcal{X}(S_t)$, which we refer to as the feasible region. Classical treatments of dynamic programming assume that we enumerate all possible elements of $\mathcal{X}(S_t)$ when solving problem (10). When x_t is a high-dimensional vector, this is again intractable.

3.1. An Approximation Strategy Using the Post-Decision State Vector

The standard version of the Bellman equation in (10) is formulated using the pre-decision state vector. If we write the Bellman equation around the post-decision state vector R^x_{t-1}, then we obtain

$$V^x_{t-1}(R^x_{t-1}) = \mathbb{E}\left\{ \max_{x_t \in \mathcal{X}(R^x_{t-1}, \hat{R}_t, \hat{D}_t)} C_t(x_t) + V^x_t(S^{M,x}(S_t, x_t)) \,\middle|\, R^x_{t-1} \right\}, \tag{11}$$

where we use the function $S^{M,x}(\cdot)$ to capture the dynamics of the post-decision state vector given in (8); that is, we have $R^x_t = S^{M,x}(S_t, x_t)$.

Not surprisingly, problem (11) is also computationally intractable. However, we can drop the expectation to write

$$\tilde{V}^x_{t-1}(R^x_{t-1}, \hat{R}_t, \hat{D}_t) = \max_{x_t \in \mathcal{X}(R^x_{t-1}, \hat{R}_t, \hat{D}_t)} C_t(x_t) + V^x_t(S^{M,x}(R^x_{t-1}, W_t(\omega), x_t)), \tag{12}$$

where $W_t(\omega) = (\hat{R}_t, \hat{D}_t)$ is a sample realization of the new information that arrived during time interval t. $\tilde{V}^x_{t-1}(S^x_{t-1}, \hat{R}_t, \hat{D}_t)$ is a place holder. Rather than computing the expectation, we solve the problem above for a particular realization of $(\hat{R}_t, \hat{D}_t)$; that is, given R^x_{t-1} and $(\hat{R}_t, \hat{D}_t)$, we compute a single decision x_t. Therefore, we can solve the second curse of dimensionality that arises due to the size of the outcome space by using the post-decision state vector.

However, we still do not know the value function $V^x_t(\cdot)$. To overcome this problem, we replace the value function with an approximation that we denote by using $\bar{V}^x_t(\cdot)$. In this case, our decision function is to solve the problem

$$X^\pi_t(R^x_{t-1}, \hat{R}_t, \hat{D}_t) = \operatorname*{argmax}_{x_t \in \mathcal{X}(R^x_{t-1}, \hat{R}_t, \hat{D}_t)} C_t(x_t) + \bar{V}^x_t(S^{M,x}(S_t, x_t)). \tag{13}$$

Therefore, we solve the first curse of dimensionality arising from the size of the state space by using approximations of the value function. Finally, we pay attention to use specially-structured value function approximations so that the problem above can be solved by using standard optimization techniques. This solves the third curse of dimensionality arising from the size of the action space.

3.2. Approximating the Value Function

Unless we are dealing with a problem with a very special structure, it is difficult to come up with good value function approximations. The approximate dynamic programming framework we propose solves problems of the form (13) for each time period t, and iteratively updates and improves the value function approximations. We describe this idea in Figure 1. We note that solving problems of the form (14) for all $t \in \mathcal{T}$ is equivalent to simulating the behavior of the policy characterized by the value function approximations $\{\bar{V}^{n-1,x}_t(\cdot) : t \in \mathcal{T}\}$. In Figure 1, we leave the structure of the value function approximations and the inner

FIGURE 1. An algorithmic framework for approximate dynamic programming.

Step 1. Choose initial value function approximations, say $\{\bar{V}_t^{0,x}(\cdot) : t \in \mathcal{T}\}$. Initialize the iteration counter by letting $n = 1$.

Step 2. Initialize the time period by letting $t = 0$. Initialize the state vector $R_0^{n,x}$ to reflect the initial state of the resources.

Step 3. Sample a realization of $(\hat{R}_t, \hat{D}_t)$, say $(\hat{R}_t^n, \hat{D}_t^n)$. Solve the problem

$$x_t^n = \underset{x_t \in \mathcal{X}(R_{t-1}^{n,x}, \hat{R}_t^n, \hat{D}_t^n)}{\operatorname{argmax}} C_t(x_t) + \bar{V}_t^{n-1,x}(S^{M,x}(S_t, x_t)) \tag{14}$$

and let $R_t^{x,n} = S^{M,x}(S_t, x_t)$.

Step 4. Increase t by 1. If $t \leq T$, then go to Step 3.

Step 5. Use the information obtained at iteration n to update the value function approximations. For the moment, we denote this by

$$\{\bar{V}_t^{n,x}(\cdot) : t \in \mathcal{T}\} = \text{Update}\big(\{\bar{V}_t^{n-1,x}(\cdot) : t \in \mathcal{T}\}, \{R_t^{n,x} : t \in \mathcal{T}\}, \{(\hat{R}_t^n, \hat{D}_t^n) : t \in \mathcal{T}\}\big),$$

where Update$(\cdot)$ can be viewed as a function that maps the value function approximations, the resource state vectors and the new information at iteration n to the updated value function approximations.

Step 6. Increase n by 1 and go to Step 2.

workings of the Update$(\cdot)$ function unspecified. Different strategies to fill in these two gaps potentially yield different approximate dynamic programming methods.

A generic structure for the value function approximations is

$$\bar{V}_t^x(R_t^x) = \sum_{f \in \mathcal{F}} \theta_{tf}\, \phi_f(R_t^x), \tag{15}$$

where $\{\phi_f(R_t^x) : f \in \mathcal{F}\}$ are often referred to as features since they capture the important characteristics of the resource state vector from the perspective of capturing the total expected profit contribution in the future. For example, if we are solving a resource allocation problem, a feature may be the number of resources with a particular attribute vector. By adjusting the parameters $\{\theta_{tf} : f \in \mathcal{F}\}$, we obtain different value function approximations. The choice of the functions $\{\phi_f(\cdot) : f \in \mathcal{F}\}$ requires some experimentation and some knowledge of the problem structure. However, for given $\{\phi_f(\cdot) : f \in \mathcal{F}\}$, there exist a variety of methods to set the values of the parameters $\{\theta_{tf} : f \in \mathcal{F}\}$ so that the value function approximation in (15) is a good approximation to the value function $V_t^x(\cdot)$.

For resource allocation problems, we further specialize the value function approximation structure in (15). In particular, we use separable value function approximations of the form

$$\bar{V}_t^x(R_t^x) = \sum_{a \in \mathcal{A}} \bar{V}_{ta}^x(R_{ta}^x), \tag{16}$$

where $\{\bar{V}_{ta}^x(\cdot) : a \in \mathcal{A}\}$ are one-dimensional functions. We focus on two cases.

1. Linear value function approximations. For these value function approximations, we have $\bar{V}_{ta}^x(R_{ta}^x) = \bar{v}_{ta}\, R_{ta}^x$, where $\bar{v}_{ta}$ are adjustable parameters. We use the notation $\{\bar{v}_{ta} : a \in \mathcal{A}\}$ for the adjustable parameters since this emphasizes that we are representing the value function approximation $\bar{V}_t^x(\cdot)$, but $\{\bar{v}_{ta} : a \in \mathcal{A}\}$ are simply different representations of $\{\theta_{tf} : f \in \mathcal{F}\}$ in (15).

2. Piecewise-linear value function approximations. These value function approximations assume that $\bar{V}_{ta}^{x}(\cdot)$ is a piecewise-linear concave function with points of nondifferentiability being subset of positive integers. In this case, letting Q be an upper bound on the total number of resources that one can have at any time period, we can characterize $\bar{V}_{ta}^{x}(\cdot)$ by a sequence of numbers $\{\bar{v}_{ta}(q) : q = 1, \ldots, Q\}$, where $\bar{v}_{ta}(q)$ is the slope of $\bar{V}_{ta}^{x}(\cdot)$ over the interval $(q-1, q)$; that is, we have $\bar{v}_{ta}(q) = \bar{V}_{ta}^{x}(q) - \bar{V}_{ta}^{x}(q-1)$. Since $\bar{V}_{ta}^{x}(\cdot)$ is concave, we have $\bar{v}_{ta}(1) \geq \bar{v}_{ta}(2) \geq \ldots \geq \bar{v}_{ta}(Q)$.

4. Monte Carlo Methods for Updating the Value Function Approximations

In this section, our goal is to propose alternatives for the Update$(\cdot)$ function in Step 5 in Figure 1.

Whether we use linear or piecewise-linear value function approximations of the form $\bar{V}_{t}^{n,x}(R_t^x) = \sum_{a \in \mathcal{A}} \bar{V}_{ta}^{n,x}(R_{ta}^x)$, each of the functions $\{\bar{V}_{ta}^{n,x}(\cdot) : a \in \mathcal{A}\}$ is characterized either by a single slope (for the linear case) or by a sequence of slopes (for the piecewise-linear case). Using e_a to denote the $|\mathcal{A}|$-dimensional unit vector with a 1 in the element corresponding to $a \in \mathcal{A}$, we would like to use $V_t^x(R_t^{n,x} + e_a) - V_t^x(R_t^{n,x})$ to update and improve the slopes that characterize the function $\bar{V}_{ta}^{n,x}(\cdot)$. However, this requires knowledge of the exact value function. Instead, letting $\tilde{V}_t^{n,x}(R_t^{n,x}, \hat{R}_t^n, \hat{D}_t^n)$ be the optimal objective value of problem (14), we propose using

$$\vartheta_{ta}^n = \tilde{V}_t^{n,x}(R_t^{n,x} + e_a, \hat{R}_t^n, \hat{D}_t^n) - \tilde{V}_t^{n,x}(R_t^{n,x}, \hat{R}_t^n, \hat{D}_t^n). \tag{17}$$

We begin by describing a possible alternative for the Update$(\cdot)$ function when the value function approximations are linear. After that, we move on to piecewise-linear value function approximations.

4.1. Updating Linear Value Function Approximations

The method that we use for updating the linear value function approximations is straightforward. Assuming that the value function approximation at iteration n is of the form $\bar{V}_{t}^{n,x}(R_t^x) = \sum_{a \in \mathcal{A}} \bar{v}_{ta}^n \, R_{ta}^x$, we let

$$\bar{v}_{ta}^n = [1 - \alpha_{n-1}] \, \bar{v}_{ta}^{n-1} + \alpha_{n-1} \, \vartheta_{ta}^n \tag{18}$$

for all $a \in \mathcal{A}$, where $\alpha_n \in [0, 1]$ is the smoothing constant at iteration n. In this case, the value function approximation to be used at iteration $n+1$ is given by $\bar{V}_{t}^{n,x}(R_t^x) = \sum_{a \in \mathcal{A}} \bar{v}_{ta}^n \, R_{ta}^x$.

Linear value function approximations can be unstable and experimental work shows that they do not perform as well as piecewise-linear value function approximations. Linear value function approximations are especially well suited to problems where the resources being managed are fairly complex producing a very large attribute space. In these problems, we typically find that R_{ta}^x is 0 or 1 and using piecewise-linear value function approximations provides little value. In addition, linear value functions are much easier to work with and generally are a good starting point.

4.2. Updating Piecewise-Linear Value Function Approximations

We now assume that the value function approximation after iteration n is of the form $\bar{V}_{t}^{n,x}(R_t^x) = \sum_{a \in \mathcal{A}} \bar{V}_{ta}^{n,x}(R_{ta}^x)$, where each $\bar{V}_{ta}^{n,x}(\cdot)$ is a piecewise-linear concave function with points of nondifferentiability being a subset of positive integers. In particular, assuming that $\bar{V}_{ta}^{n,x}(0) = 0$ without loss of generality, we represent $\bar{V}_{ta}^{n,x}(\cdot)$ by a sequence of slopes $\{\bar{v}_{ta}^n(q) : q = 1, \ldots, Q\}$ as in Section 3.2, where we have $\bar{v}_{ta}^n(q) = \bar{V}_{ta}^{n,x}(q) - \bar{V}_{ta}^{n,x}(q-1)$. Concavity of $\bar{V}_{ta}^{n,x}(\cdot)$ implies that $\bar{v}_{ta}^n(1) \geq \bar{v}_{ta}^n(2) \geq \ldots \geq \bar{v}_{ta}^n(Q)$.

We update $\bar{V}_{ta}^{n,x}(\cdot)$ by letting

$$\theta_{ta}^{n}(q) = \begin{cases} [1-\alpha_{n-1}]\,\bar{v}_{ta}^{n-1}(q) + \alpha_{n-1}\,\vartheta_{ta}^{n} & \text{if } q = R_{ta}^{n,x}+1 \\ \bar{v}_{ta}^{n-1}(q) & \text{if } q \in \{1,\ldots,R_{ta}^{n,x}, R_{ta}^{n,x}+2,\ldots,Q\}. \end{cases} \tag{19}$$

The expression above is similar to (18), but the smoothing operation applies only to the "relevant" part of the domain of $\bar{V}_{ta}^{n,x}(\cdot)$. However, we note that we may not have $\theta_{ta}^{n}(1) \geq \theta_{ta}^{n}(2) \geq \ldots \geq \theta_{ta}^{n}(Q)$, which implies that if we let $\bar{V}_{ta}^{n,x}(\cdot)$ be the piecewise-linear function characterized by the sequence of slopes $\theta_{ta}^{n} = \{\theta_{ta}^{n}(q) : q = 1,\ldots,Q\}$, then $\bar{V}_{ta}^{n,x}(\cdot)$ is not necessarily concave. To make sure that $\bar{V}_{ta}^{n,x}(\cdot)$ is concave, we choose a sequence of slopes $\bar{v}_{ta}^{n} = \{\bar{v}_{ta}^{n}(q) : q = 1,\ldots,Q\}$ such that $\bar{v}_{ta}^{n}$ and θ_{ta}^{n} are not too "far" from each other and the sequence of slopes $\bar{v}_{ta}^{n}$ satisfy $\bar{v}_{ta}^{n}(1) \geq \bar{v}_{ta}^{n}(2) \geq \ldots \geq \bar{v}_{ta}^{n}(Q)$. In this case, we let $\bar{V}_{ta}^{n,x}(\cdot)$ be the piecewise-linear concave function characterized by the sequence of slopes $\bar{v}_{ta}^{n}$.

There are several methods for choosing the sequence of slopes $\{\bar{v}_{ta}^{n+1}(q) : q = 1,\ldots,Q\}$. One possible method is to let $\bar{v}_{ta}^{n}$ be as follows

$$\begin{aligned} \bar{v}_{ta}^{n} = \text{argmin} \quad & \sum_{q=1}^{Q} \left[z_q - \theta_{ta}^{n}(q)\right]^2 \\ \text{subject to} \quad & z_{q-1} - z_q \geq 0 \qquad \text{for all } q = 2,\ldots,Q. \end{aligned} \tag{20}$$

Therefore, this method chooses the vector $\bar{v}_{ta}^{n}$ as the projection of the vector θ_{ta}^{n} onto the set $\mathcal{W} = \{z \in \mathbb{R}^Q : z_1 \geq z_2 \geq \ldots \geq z_Q\}$; that is, we have

$$\bar{v}_{ta}^{n} = \underset{z \in \mathcal{W}}{\text{argmin}} \quad \|z - \theta_{ta}^{n}\|_2. \tag{21}$$

Using the Karush-Kuhn-Tucker conditions for problem (20), we can come up with a closed-form expression for the projection in (21). We only state the final result here. Since the vector θ_{ta}^{n} differs from the vector $\bar{v}_{ta}^{n}$ in one component and we have $\bar{v}_{ta}^{n}(1) \geq \bar{v}_{ta}^{n}(2) \geq \ldots \geq \bar{v}_{ta}^{n}(Q)$, there are three possible cases to consider; either $\theta_{ta}^{n}(1) \geq \theta_{ta}^{n}(2) \geq \ldots \geq \theta_{ta}^{n}(Q)$, or $\theta_{ta}^{n}(R_{ta}^{n,x}) < \theta_{ta}^{n}(R_{ta}^{n,x}+1)$, or $\theta_{ta}^{n}(R_{ta}^{n,x}+1) < \theta_{ta}^{n}(R_{ta}^{n,x}+2)$ should hold. If the first case holds, then we can choose $\bar{v}_{ta}^{n+1}$ in (21) as θ_{ta}^{n} and we are done. If the second case holds, then we find the largest $q^* \in \{2,\ldots,R_{ta}^{n,x}+1\}$ such that

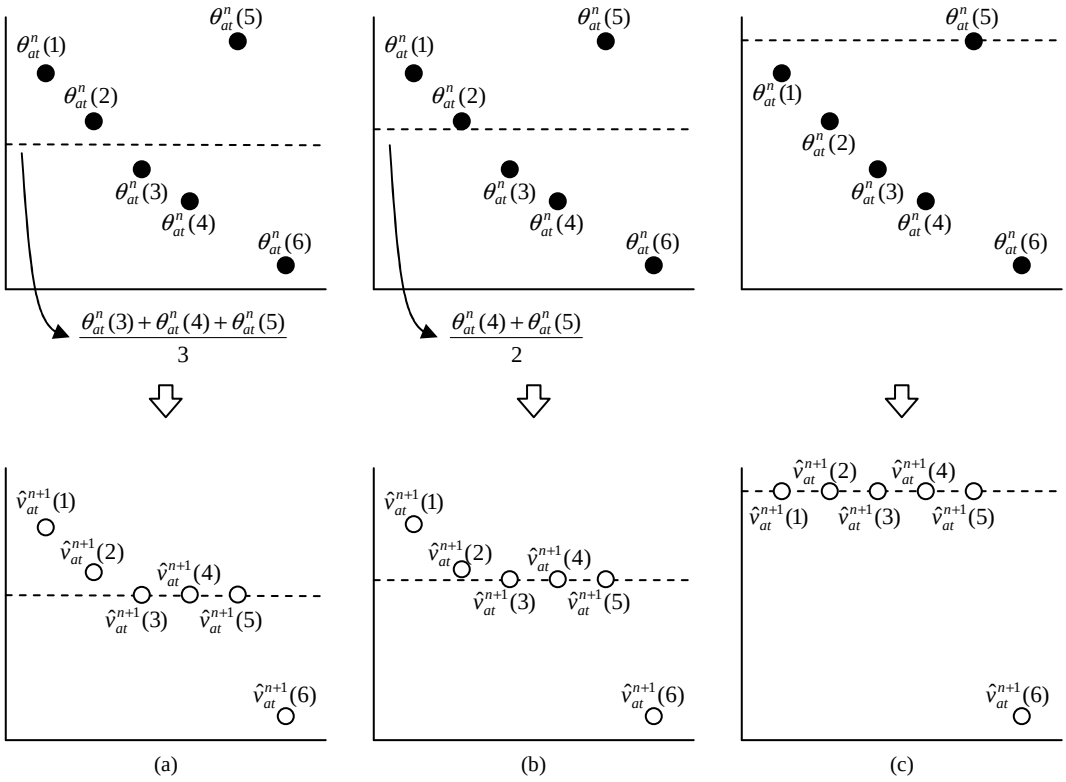
$$\theta_{ta}^{n}(q^*-1) \geq \frac{1}{R_{ta}^{n,x}+2-q^*} \sum_{q=q^*}^{R_{ta}^{n,x}+1} \theta_{ta}^{n}(q).$$

If such q^* cannot be found, then we let $q^* = 1$. It is straightforward to check that the vector $\bar{v}_{ta}^{n}$ given by

$$\bar{v}_{ta}^{n}(q) = \begin{cases} \dfrac{1}{R_{ta}^{n,x}+2-q^*} \displaystyle\sum_{q=q^*}^{R_{ta}^{n,x}+1} \theta_{ta}^{n}(q) & \text{if } q \in \{q^*,\ldots,R_{ta}^{n,x}+1\} \\ \theta_{ta}^{n}(q) & \text{if } q \notin \{q^*,\ldots,R_{ta}^{n,x}+1\} \end{cases} \tag{22}$$

satisfies the Karush-Kuhn-Tucker conditions for problem (20). If the third case holds, then one can apply a similar argument. Figure 2.a shows how this method works. The black circles in the top portion of this figure show the sequence of slopes $\{\theta_{ta}^{n}(q) : q = 1,\ldots,Q\}$, whereas the white circles in the bottom portion show the sequence of slopes $\{\bar{v}_{ta}^{n+1}(q) : q = 1,\ldots,Q\}$ computed through (22).

FIGURE 2. Three possible methods for choosing the vector $\bar{v}_{ta}^{n+1}$.



Note. In this figure, we assume that $Q=6$, $R_{ta}^{n,x}+1=5$ and $q^*=3$.

Recalling the three possible cases considered above, a second possible method first computes

$$M^* = \begin{cases} \theta_{ta}^n(R_{ta}^{n,x}+1) & \text{if } \theta_{ta}^n(1) \geq \theta_{ta}^n(2) \geq \ldots \geq \theta_{ta}^n(Q) \\ \dfrac{\theta_{ta}^n(R_{ta}^{n,x}) + \theta_{ta}^n(R_{ta}^{n,x}+1)}{2} & \text{if } \theta_{ta}^n(R_{ta}^{n,x}) < \theta_{ta}^n(R_{ta}^{n,x}+1) \\ \dfrac{\theta_{ta}^n(R_{ta}^{n,x}+1) + \theta_{ta}^n(R_{ta}^{n,x}+2)}{2} & \text{if } \theta_{ta}^n(R_{ta}^{n,x}+1) < \theta_{ta}^n(R_{ta}^{n,x}+2), \end{cases} \tag{23}$$

and lets

$$\bar{v}_{ta}^n(q) = \begin{cases} \max\left\{\theta_{ta}^n(q),\, M^*\right\} & \text{if } q \in \{1,\ldots,R_{ta}^{n,x}\} \\ M^* & \text{if } q = R_{ta}^{n,x}+1 \\ \min\left\{\theta_{ta}^n(q),\, M^*\right\} & \text{if } q \in \{R_{ta}^{n,x}+2,\ldots,Q\}. \end{cases} \tag{24}$$

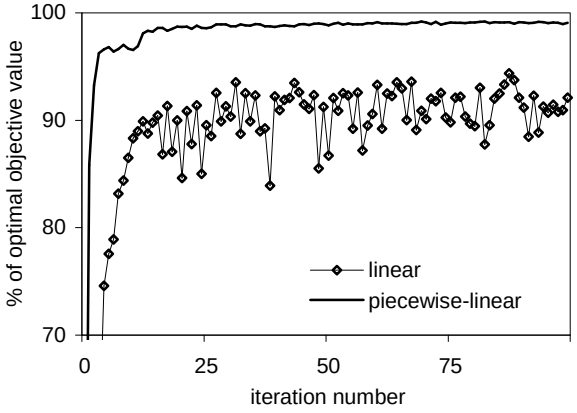
Interestingly, it can be shown that (23) and (24) are equivalent to letting

$$\bar{v}_{ta}^{n+1} = \operatorname*{argmin}_{z \in \mathcal{W}} \; \|z - \theta_{ta}^n\|_\infty.$$

Therefore, the first method is based on a Euclidean-norm projection, whereas the second method is based on a max-norm projection. Figure 2.b shows how this method works.

A slight variation on the second method yields a third method, which computes $M^* = \theta_{ta}^n(R_{ta}^{n,x}+1)$ and lets the vector $\bar{v}_{ta}^{n+1}$ be as in (24). This method does not have an interpretation as a projection. Figure 2.c shows how this method works.

FIGURE 3. Performances of linear and piecewise-linear value function approximations on a resource allocation problem with deterministic data.



There are convergence results for the three methods described above. All of these results are in limited settings that assume that the planning horizon contains two time periods and the state vector is one-dimensional. Roughly speaking, they show that if the state vector $R_1^{n,x}$ generated by the algorithmic framework in Figure 1 satisfies $\sum_{n=1}^{\infty} \mathbf{1}(R_1^{n,x} = q) = \infty$ with probability 1 for all $q = 1, \ldots, Q$ and we use one of the three methods described above to update the piecewise-linear value function approximations, then we have $\lim_{n\to\infty} \bar{v}_1^n(R_1^x) = V_1(R_1^x) - V_1(R_1^x - 1)$ for all $R_1^x = 1, \ldots, Q$ with probability 1. Throughout, we omit the subscript a because the state vector is one-dimensional and use $\mathbf{1}(\cdot)$ to denote the indicator function. When we apply these methods to large resource allocation problems with multi-dimensional state vectors, they are only approximate methods that seem to perform quite well in practice.

Experimental work indicates that piecewise-linear value function approximations can provide better objective values and more stable behavior than linear value function approximations. Figure 3 shows the performances of linear and piecewise-linear value function approximations on a resource allocation problem with deterministic data. The horizontal axis is the iteration number in the algorithmic framework in Figure 1. The vertical axis is the performance of the policy that is obtained at a particular iteration, expressed as a percentage of the optimal objective value. We obtain the optimal objective value by formulating the problem as a large integer program. Figure 3 shows that the policies characterized by piecewise-linear value function approximations may perform almost as well as the optimal solution, whereas the policies characterized by linear value function approximations lag behind significantly. Furthermore, the performances of the policies characterized by linear value function approximations at different iterations can fluctuate. Nevertheless, linear value function approximations may be used as prototypes before moving on to more sophisticated approximation strategies or we may have to live with them simply because the resource allocation problem we are dealing with is too complex.

5. Stepsizes

Approximate dynamic programming depends heavily on using information from the latest iteration to update a value function approximation. This results in updates of the form

$$\bar{v}_{ta}^n = [1 - \alpha_{n-1}]\, \bar{v}_{ta}^{n-1} + \alpha_{n-1}\, \vartheta_{ta}^n, \tag{25}$$

where α_{n-1} is the stepsize used in iteration n. This intuitive updating formula is known variously as exponential smoothing, a linear filter or a stochastic approximation procedure. The equation actually comes from the optimization problem

$$\min_{\theta} \mathbb{E}\{F(\theta, \hat{R})\},$$

where $F(\theta, \hat{R})$ is a function of θ and random variable $\hat{R}$. Further, we assume that we cannot compute the expectation either because the function is too complicated or because we do not know the distribution of $\hat{R}$. We can still solve the problem using an algorithm of the form

$$\theta^n = \theta^{n-1} - \alpha_{n-1} \nabla F(\theta^{n-1}, \hat{R}^n), \tag{26}$$

where θ^{n-1} is our estimate of the optimal solution after iteration $n-1$ and $\hat{R}^n$ is a sample of the random variable $\hat{R}$ at iteration n. If $F(\cdot, \hat{R}^n)$ is not differentiable, then we assume that $\nabla F(\theta^{n-1}, \hat{R}^n)$ is a subgradient of the function. The updating in equation (26) is known as a stochastic gradient algorithm, since we are taking a gradient of $F(\cdot, \hat{R}^n)$ with respect to θ at a sample realization of the random variable $\hat{R}$.

Assume that our problem is to estimate the mean of the random variable $\hat{R}$. We assume that the distribution of the random variable $\hat{R}$ is unknown, but we can obtain samples $\hat{R}^1, \hat{R}^2, \ldots$. Since we have $\mathbb{E}\{\hat{R}\} = \text{argmin}_\theta \mathbb{E}\{(\theta - \hat{R})^2\}$, a reasonable approach is to let

$$F(\theta, \hat{R}) = \frac{1}{2}(\theta - \hat{R})^2$$

and use (26). Letting θ^n be the estimate of $\mathbb{E}\{\hat{R}\}$ obtained after iteration n, since we have $\nabla F(\theta, \hat{R}) = (\theta - \hat{R})$, we obtain

$$\begin{aligned} \theta^n &= \theta^{n-1} - \alpha_{n-1} \nabla F(\theta^{n-1}, \hat{R}^n) \\ &= \theta^{n-1} - \alpha_{n-1} (\theta^{n-1} - \hat{R}^n) = (1 - \alpha_{n-1}) \theta^{n-1} + \alpha_{n-1} \hat{R}^n. \end{aligned}$$

Among the last two equalities above, the first one has the same form as the stochastic gradient algorithm and the second one has the same form as exponential smoothing.

There is an elegant theory that tells us that this method works, but there are some simple restrictions on the stepsizes. In addition to the requirement that $\alpha_{n-1} \geq 0$ for $n = 1, 2, \ldots$, the stepsizes must also satisfy

$$\sum_{n=1}^{\infty} \alpha_{n-1} = \infty \qquad \sum_{n=1}^{\infty} (\alpha_{n-1})^2 < \infty.$$

The first condition ensures that the stepsizes do not decline too quickly; otherwise the algorithm may stall out prematurely. The second ensures that they do not decline too slowly, which ensures that the algorithm actually converges in the limit. One stepsize rule that satisfies this condition is $\alpha_{n-1} = 1/(n-1)$. This rule is special because it produces a simple averaging of all the observations, which is to say that

$$\theta^n = \frac{1}{n} \sum_{m=1}^{n} \hat{R}^m.$$

If we are getting a series of observations of $\hat{R}$ from a stationary distribution, this would be fine; in fact, this is the best we can do. However, in dynamic programming, our updates of the value function are changing over the iterations as we try to converge on an optimal policy. As a result, the values ϑ_{ta}^n are coming from a distribution that is changing over the

iterations. For this reason, it is well known that the so-called "$1/n$" stepsize rule produces stepsizes that decline much too quickly.

A variety of strategies have evolved over the years to counter this effect. One fairly general class of formulas is captured by

$$\alpha_n = \begin{cases} \alpha_0 & \text{if } n = 0 \\ \alpha_0 \frac{\left(\frac{b}{n}+a\right)}{\left(\frac{b}{n}+a+n^\beta-1\right)} & \text{if } n > 0. \end{cases}$$

If $b = 0$, $\alpha_0 = 1$, $\beta = 1$ and $a = 1$, then we obtain the "$1/n$" stepsize rule. As a is increased (values in the 5 to 20 range work quite well) or β is decreased (for theoretical reasons, it should stay above 0.5), the rate at which the stepsize decreases slows quite a bit. Raising the parameter b has the effect of keeping the stepsize very close to the initial value for a while before allowing the stepsize to decrease. This is useful for certain classes of delayed learning, where a number of iterations must occur before the system starts to obtain meaningful results. We have found that $a = 8$, $b = 0$ and $\beta = 0.7$ works quite well for many dynamic programming applications.

Another useful rule is McClain's formula, given by

$$\alpha_n = \begin{cases} \alpha_0 & \text{if } n = 0 \\ \frac{\alpha_{n-1}}{1+\alpha_{n-1}-\bar{\alpha}} & \text{if } n \geq 1. \end{cases}$$

If $\bar{\alpha} = 0$ and $\alpha_0 = 1$, then this formula gives $\alpha_n = 1/n$. For $0 < \bar{\alpha} < 1$, the formula produces a sequence of decreasing stepsizes that initially behaves like $1/n$, but decreases to $\bar{\alpha}$ instead of 0. This is a way of ensuring that the stepsize does not get too small.

The challenge with stepsizes is that if we are not careful, then we may design an algorithm that works poorly when, in fact, the only problem is the stepsize. It may be quite frustrating tuning the parameters of a stepsize formula; we may be estimating many thousands of parameters and the best stepsize formula may be different for each parameter.

For this reason, researchers have studied a number of stochastic stepsize formulas. These are stepsize rules where the size of the stepsize depends on what is happening over the course of the algorithm. Since the stepsize at iteration n depends on the data, the stepsize itself is a random variable. One of the earliest and most famous of the stochastic stepsize rules is known as Kesten's rule given by

$$\alpha_n = \alpha_0 \frac{a}{a + K^n}, \tag{27}$$

where α_0 is the initial stepsize and a is a parameter to be calibrated. Letting

$$\varepsilon^n = \theta^{n-1} - \hat{R}^n$$

be the error between our previous estimate of the random variable and the latest observation, if θ^{n-1} is far from the true value, then we expect to see a series of errors with the same sign. The variable K^n counts the number of times that the sign of the error has changed by

$$K^n = \begin{cases} n & \text{if } n = 0, 1 \\ K^{n-1} + \mathbf{1}(\varepsilon^n \varepsilon^{n-1} < 0) & \text{otherwise.} \end{cases} \tag{28}$$

Thus, every time the sign changes, indicating that we are close to the optimal solution, the stepsize decreases.

Ideally, a stepsize formula should decline as the level of variability in the observations increase and should increase when the underlying signal is changing quickly. A formula that does this is

$$\alpha^n = 1 - \frac{\sigma^2}{\left(1 + \lambda^{n-1}\right)\sigma^2 + \left(\beta^n\right)^2},$$

FIGURE 4. The optimal stepsize algorithm.

Step 0. Choose an initial estimate $\bar{\theta}^0$ and an initial stepsize α_0. Assign initial values to the parameters by letting $\bar{\beta}^0 = 0$ and $\bar{\delta}^0 = 0$. Choose an initial value for the error stepsize γ_0 and a target value for the error stepsize $\bar{\gamma}$. Set the iteration counter $n = 1$.

Step 1. Obtain the new observation $\hat{R}^n$.

Step 2. Update the following parameters by letting

$$\begin{aligned}
\gamma_n &= \frac{\gamma_{n-1}}{1+\gamma_{n-1}-\bar{\gamma}} \\
\bar{\beta}^n &= (1-\gamma_n)\,\bar{\beta}^{n-1} + \gamma_n(\hat{R}^n - \bar{\theta}^{n-1}) \\
\bar{\delta}^n &= (1-\gamma_n)\,\bar{\delta}^{n-1} + \gamma_n(\hat{R}^n - \bar{\theta}^{n-1})^2 \\
(\bar{\sigma}^n)^2 &= \frac{\bar{\delta}^n - (\bar{\beta}^n)^2}{1+\bar{\lambda}^{n-1}}.
\end{aligned}$$

Step 3. If $n > 1$, then evaluate the stepsizes for the current iteration by

$$\alpha_n = 1 - \frac{(\bar{\sigma}^n)^2}{\bar{\delta}^n}.$$

Step 4. Update the coefficient for the variance of the smoothed estimate by

$$\bar{\lambda}^n = \begin{cases} (\alpha_n)^2 & \text{if } n = 1 \\ (1-\alpha_n)^2\bar{\lambda}^{n-1} + (\alpha_n)^2 & \text{if } n > 1. \end{cases}$$

Step 5. Smooth the estimate by

$$\bar{\theta}^n = (1-\alpha_{n-1})\,\bar{\theta}^{n-1} + \alpha_{n-1}\,\hat{R}^n.$$

Step 6. If $\bar{\theta}^n$ satisfies some termination criterion, then stop. Otherwise, set $n = n+1$ and go to Step 1.

---

where

$$\lambda^n = \begin{cases} (\alpha_n)^2 & \text{if } n = 1 \\ (\alpha_n)^2 + (1-\alpha_n)^2\lambda^{n-1} & \text{if } n > 1. \end{cases}$$

In the expression above, σ^2 is the noise in the observations and β^n is the difference between the true value and the estimated value, which we refer to as the bias. It can be shown that if $\sigma^2 = 0$, then $\alpha_n = 1$, whereas if $\beta^n = 0$, then $\alpha_n = 1/n$. The problem is that neither of these quantities would normally be known; in particular, if we knew the bias, then it means we know the true value function.

Figure 4 presents an adaptation of this formula for the case where the noise and bias are not known. This formula has been found to provide consistently good results for a broad range of problems, including those with delayed learning.

6. Other Approaches for Dynamic Resource Allocation Problems

To understand the relative simplicity of approximate dynamic programming and to provide benchmarks to measure solution quality, it is useful to review other methods for solving resource allocation problems.

6.1. A Deterministic Model

A common strategy to deal with randomness is to assume that the future random quantities take on their expected values and to formulate a deterministic optimization problem. For the resource allocation setting, this problem takes the form

$$\begin{aligned} \max \quad & \sum_{a\in\mathcal{A}}\sum_{d\in\mathcal{D}} c_{0ad}\, x_{0ad} && && (29)\\ \text{subject to} \quad & \sum_{d\in\mathcal{D}} x_{0ad} = R_{0a} && \text{for all } a\in\mathcal{A} \\ & -\sum_{a'\in\mathcal{A}}\sum_{d\in\mathcal{D}} \delta_a(a'd)\, x_{0a'd} + \sum_{d\in\mathcal{D}} x_{0ad} = \mathbb{E}\{\hat{R}_{ta}\} && \text{for all } a\in\mathcal{A} \\ & x_{0ad}\in\mathbb{Z}_+ && \text{for all } a\in\mathcal{A},\ d\in\mathcal{D}, \end{aligned}$$

It is important to keep in mind that the time at which flows happen is imbedded in the attribute vector. This makes for a very compact model, but one which is less transparent. In practice, we use problem (29) on a rolling horizon basis; we solve this problem to make the decisions at the first time period and implement these decisions. When it is time to make the decisions at the second time period, we solve a similar problem that involves the known resource state vector and the demands at the second time period.

Problem (29) uses only the expected values of the random quantities, disregarding the distribution information. However, there are certain applications, such as airline fleet assignment, where the uncertainty does not play a crucial role and problem (29) can efficiently be solved as an integer multicommodity min-cost network flow problem.

6.2. Scenario-Based Stochastic Programming Methods

Stochastic programming emerges as a possible approach when one attempts to use the distribution information. In the remainder of this section, we review stochastic programming methods applicable to resource allocation problems. Thus far, we mostly focused on problems where the decision variables take integer values. There has been much progress in the area of integer stochastic programming within the last decade, but there does not exist integer stochastic programming methods to our knowledge that can solve the resource allocation problems in the full generality that we present here. For this reason, we relax the integrality constraints throughout this section. To make the ideas transparent, we assume that the planning horizon contains two time periods, although most of the methods apply to problems with longer planning horizons.

Scenario-based stochastic programming methods assume that there exist a finite set of possible realizations for the random vector $(\hat{R}_1, \hat{D}_1)$, which we denote by $\{(\hat{R}_1(\omega), \hat{D}_1(\omega)) : \omega\in\Omega\}$. In this case, using $p(\omega)$ to denote the probability of realization $(\hat{R}_1(\omega), \hat{D}_1(\omega))$, the exact value function at the second time period can be computed by solving

$$\begin{aligned} V_0(R_0^x) = \max \quad & \sum_{\omega\in\Omega}\sum_{a\in\mathcal{A}}\sum_{d\in\mathcal{D}} p(\omega)\, c_{1ad}\, x_{1ad}(\omega) && && (30)\\ \text{subject to} \quad & \sum_{d\in\mathcal{D}} x_{1ad}(\omega) = R_{0a}^x + \hat{R}_{1a}(\omega) && \text{for all } a\in\mathcal{A},\ \omega\in\Omega && (31)\\ & \sum_{a\in\mathcal{A}} x_{1ad} \le \hat{D}_{1,b_d}(\omega) && \text{for all } d\in\mathcal{D}^D,\ \omega\in\Omega, \end{aligned}$$

where we omit the nonnegativity constraints for brevity. This approach allows complete generality in the correlation structure among the elements of the random vector $(\hat{R}_1, \hat{D}_1)$, but it assumes that this random vector is independent of R_1. Since the decision variables are $\{x_{1ad}(\omega) : a\in\mathcal{A},\ d\in\mathcal{D},\ \omega\in\Omega\}$, problem (30) can be large for practical applications.

6.3. Benders Decomposition-Based Methods

Since the resource state vector R_0^x appears on the right side of constraints (31), $V_0(R_0^x)$ is a piecewise-linear concave function of R_0^x. Benders decomposition-based methods refer to a class of methods that approximate the exact value function $V_0(\cdot)$ by a series of cuts that are constructed iteratively. In particular, letting $\{\lambda_1^i : i = 1, \ldots, n-1\}$ and $\{\beta_{1a}^i : a \in \mathcal{A},\ i = 1, \ldots, n-1\}$ be the sets of coefficients characterizing the cuts that have been constructed up to iteration n, the function

$$\bar{V}_0^n(R_0^x) = \min_{i \in \{1,\ldots,n-1\}} \lambda_1^i + \sum_{a \in \mathcal{A}} \beta_{1a}^i \, R_{0a}^x \tag{32}$$

is the approximation to the exact value function $V_0(\cdot)$ at iteration n. The details of how to generate the cuts are beyond our presentation.

6.4. Auxiliary Functions

As a last possible stochastic programming method, we describe an algorithm called the stochastic hybrid approximation procedure (SHAPE). This method is similar to the methods described in Section 4; it iteratively updates an approximation to the value function by using a formula similar to (18).

SHAPE uses value function approximations of the form

$$\bar{V}_0^{n,x}(R_0^x) = \bar{W}_0(R_0^x) + \sum_{a \in \mathcal{A}} \bar{v}_{0a}^n \, R_{0a}^x, \tag{33}$$

where $\bar{W}_0(\cdot)$ is a function specified in advance. In general, $\bar{W}_0(\cdot)$ is chosen so that it is easy to work with; for example, a polynomial. However, the procedure works best when $\bar{W}_0(\cdot)$ approximately captures the general shape of the value function. The second term on the right side of (33) is a linear value function approximation component that is adjusted iteratively. Consequently, the first nonlinear component of the value function approximation does not change over the iterations, but the second linear component is adjustable. We assume that $\bar{W}_0(\cdot)$ is a differentiable concave function with the gradient $\nabla \bar{W}_0(R_0^x) = (\nabla_a \bar{W}_0(R_0^x))_{a \in \mathcal{A}}$.

Using the value function approximation in (33), we first solve the approximate subproblem at the first time period to obtain

$$x_0^n = \operatorname*{argmax}_{x_0 \in \mathcal{X}(R_0)} C_0(x_0) + \bar{V}_0^{n-1,x}(R^{M,x}(R_0, x_0)). \tag{34}$$

Letting $R_0^{n,x} = S^{M,x}(S_0, x_0^n)$ and $(\hat{R}_1^n, \hat{D}_1^n)$ be a sample of $(\hat{R}_1, \hat{D}_1)$, we then solve

$$\operatorname*{argmax}_{x_1 \in \mathcal{X}(R_0^{n,x}, \hat{R}_1^n, \hat{D}_1^n)} C_1(x_1).$$

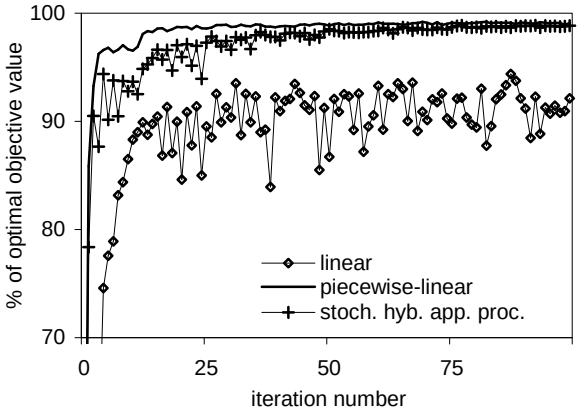
In this case, using $\{\pi_{1a}^n : a \in \mathcal{A}\}$ to denote the optimal values of the dual variables associated with constraints (4) in the problem above, we let

$$\bar{v}_{0a}^n = [1 - \alpha_{n-1}] \, \bar{v}_{0a}^{n-1} + \alpha_{n-1} \, [\pi_{1a}^n - \nabla_a \bar{V}_0^{n-1}(R_0, x_0)],$$

where $\alpha_{n-1} \in [0, 1]$ is the smoothing constant at iteration n. Therefore, the value function approximation at iteration n is given by $\bar{V}_0^{n,x}(R_0^x) = \bar{W}_0(R_0^x) + \sum_{a \in \mathcal{A}} \bar{v}_{0a}^n \, R_{0a}^x$. It is possible to show that this algorithm produces the optimal solution for two-period problems.

This method is simple to implement. Since we only update the linear component of the value function approximation, the structural properties of the value function approximation do not change. For example, if we choose $\bar{W}_0(\cdot)$ as a separable quadratic function, then the value function approximation is a separable quadratic function at every iteration. Nevertheless, SHAPE has not seen much attention from the perspective of practical implementations.

FIGURE 5. Performances of SHAPE, and linear and piecewise-linear value function approximations.



The first reason for this is that $\bar{V}_0^{n,x}(\cdot)$ is a differentiable function and the approximate subproblem in (34) is a smooth optimization problem. Given the surge in quadratic programming packages, we do not think this is a major issue any more. The second reason is that the practical performance of the procedure can depend on the choice of $\bar{W}_0(\cdot)$ and there is no clear guideline for this choice. We believe that the methods described in Section 4 can be used for this purpose. We can use these methods to construct a piecewise-linear value function approximation, fit a strongly separable quadratic function to the piecewise-linear value function approximation and use this fitted function for $\bar{W}_0(\cdot)$.

Figure 5 shows the performances of SHAPE, linear value function approximations and piecewise-linear value function approximations on a resource allocation problem with deterministic data. The objective values obtained by SHAPE at the early iterations fluctuate but they quickly stabilize, whereas the objective values obtained by linear value function approximations continue to fluctuate. The concave "auxiliary" function that SHAPE uses prevents the "bang-bang" behavior of linear value function approximations and provides more stable performance.

7. Computational Results

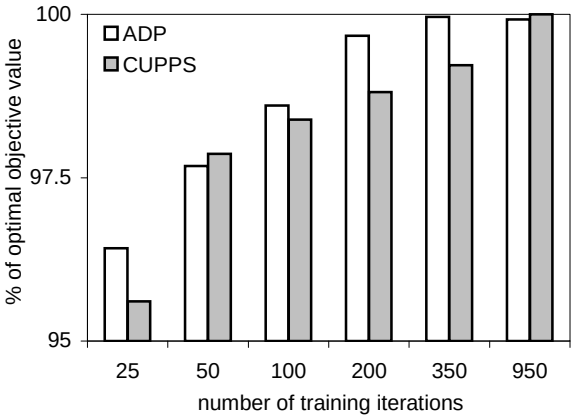
This section presents computational experiments on a variety of resource allocation problems. We begin by considering two-period problems and move on to multiple-period problems later. The primary reason that we consider two-period problems is that there exist a variety of solution methods for them, some of which are described in Section 6, that we can use as benchmarks. This gives us a chance to carefully test the performance of the algorithmic framework in Figure 1.

7.1. Two-Period Problems

In this section, we present computational experiments on two-period problems arising from the fleet management setting. We assume that there is a single vehicle type and it takes one time period to move between any origin-destination pair. In this case, the attribute vector in (1) is of the form $a = [\text{inbound/current location}]$ and the attribute space $\mathcal{A}$ is simply the set of locations in the transportation network. There are two decision types with $\mathcal{C} = \{D, M\}$, where $\mathcal{D}^D$ and $\mathcal{D}^M$ have the same interpretations as in Section 2.5. We use piecewise-linear value function approximations and update them by using (19) and (20) with $\alpha_n = 20/(40+n)$.

We generate a certain number of locations over a 100×100 region. At the beginning of the planning horizon, we spread the fleet uniformly over these locations. The loads between

FIGURE 6. Performances of ADP and CUPPS for different numbers of training iterations.



different origin-destination pairs and at different time periods are sampled from the Poisson distributions with the appropriate means. We pay attention to work on problems where the number of loads that are inbound to a particular location is negatively correlated with the number of loads that are outbound from that location. We expect that these problems require plenty of empty repositioning movements in their optimal solutions and naive methods should not provide good solutions for them.

Evaluating the performances of the methods presented in this chapter requires two sets of iterations. In the first set of iterations, which we refer to as the training iterations, we follow the algorithmic framework in Figure 1; we sample a realization of the random vector $(\hat{R}_t, \hat{D}_t)$ and solve problem (14) for each time period t, and update the value function approximations. In the second set of iterations, which we refer to as the testing iterations, we fix the value function approximations and simply simulate the behavior of the policy characterized by the value function approximations that are obtained during the training iterations. Consequently, the goal of the testing iterations is to test the quality of the value function approximations. For Benders decomposition-based methods, the training iterations construct the cuts that approximate the value functions, whereas the testing iterations simulate the behavior of the policy characterized by the cuts that are constructed during the training iterations. We vary the number of training iterations to see how fast we can obtain good policies through different methods. The particular version of Benders decomposition-based method that we use in our computational experiments is called cutting plane and partial sampling method. We henceforth refer to the approximate dynamic programming framework in Figure 1 as ADP and cutting plane and partial sampling method as CUPPS.

For a test problem that involves 30 locations, Figure 6 shows the average objective values obtained in the testing iterations as a function of the number of training iterations. The white and gray bars in this figure respectively correspond to ADP and CUPPS. When the number of training iterations is relatively small, it appears that ADP provides better objective values than CUPPS. Since CUPPS eventually solves the problem exactly and ADP is only an approximation strategy, if the number of training iterations is large, then CUPPS provides better objective values than ADP. Even after CUPPS obtains the optimal solution, the performance gap between ADP and CUPPS is a fraction of a percent. Furthermore, letting $\{\bar{V}_t^{n,x}(\cdot) : t \in \mathcal{T}\}$ be the set of value function approximations obtained by ADP at iteration n, Figure 7 shows the performance of the policy characterized by the value function approximations $\{\bar{V}_t^{n,x}(\cdot) : t \in \mathcal{T}\}$ as a function of the iteration number n. Performances of the policies stabilize after about 1500 training iterations.

For test problems that involve different numbers of locations, Figure 8 shows the average objective values obtained in the testing iterations. In this figure, the number of training

FIGURE 7. Performances of the policies obtained by ADP as a function of the number of training iterations.

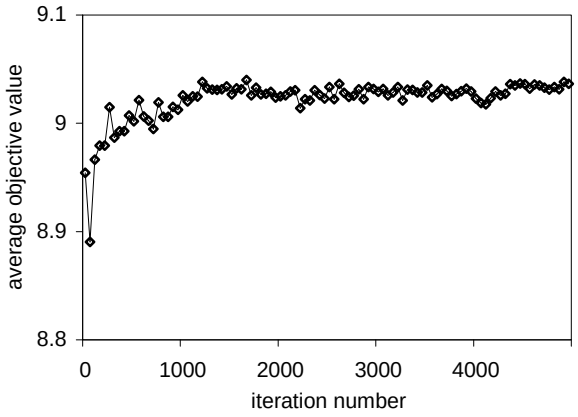
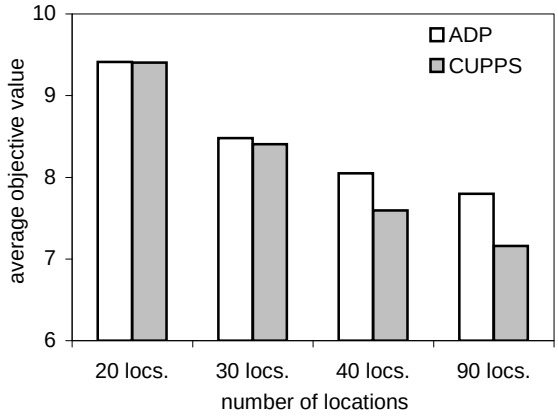


FIGURE 8. Performances of ADP and CUPPS for problems with different numbers of locations.



iterations is fixed at 200. For problems with small numbers of locations, the objective values obtained by ADP and CUPPS are very similar. As the number of locations grows, the objective values obtained by ADP are noticeably better than those obtained by CUPPS. The number of locations gives the number of dimensions of the value function. Therefore, for problems that involve high-dimensional value functions, it appears that ADP obtains good policies faster than CUPPS.

7.2. Multi-Period Problems

This section presents computational experiments on multi-period problems arising from the fleet management setting. To introduce some variety, we now assume that there are multiple vehicle and load types. In this case, the attribute space of the resources consists of vectors of the form (1). We assume that we obtain a profit of $r\,D(o,d)\,C(l,v)$ when we use a vehicle of type v to carry a load of type l from location o to d, where r is the profit per mile, $D(o,d)$ is the distance between origin-destination pair (o,d) and $C(l,v) \in [0,1]$ captures the compatibility between load type l and vehicle type v. As $C(l,v)$ approaches 0, load type l and vehicle type v become less compatible. We use piecewise-linear value function approximations and update them by using (19) and (20) with $\alpha_n = 20/(40+n)$.

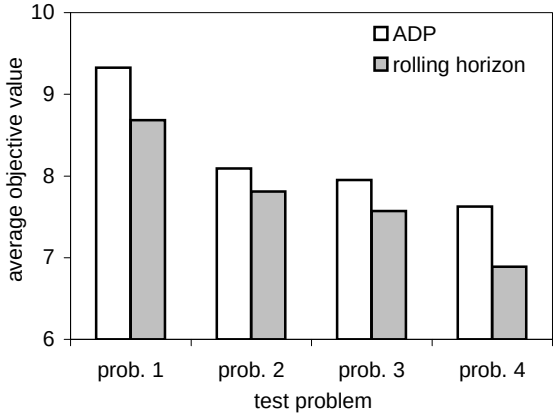
We begin by exploring the performance of ADP on problems where $\{(\hat{R}_t, \hat{D}_t) : t \in \mathcal{T}\}$ are deterministic. These problems can be formulated as integer multicommodity min-cost network flow problems as in problem (29); we solve their linear programming relaxations

TABLE 1. Performance of ADP on different test problems.

Problem	(20,60,200)	(20,30,200)	(20,90,200)	(10,60,200)	(40,60,200)	(20,60,100)	(20,60,400)
% of opt. obj.val.	99.5	99.7	99.3	99.8	99.0	97.2	99.5

Note. The triplets denote the characteristics of the test problems, where the three elements are the number of locations, the number of time periods and the fleet size.

FIGURE 9. Performances of ADP and the rolling horizon strategy on different test problems.



to obtain upper bounds on the optimal objective values. Table 1 shows the ratios of the objective values obtained by ADP and by the linear programming relaxations. ADP obtains objective values that are within 3% of the upper bounds on the optimal objective values.

We use the so-called rolling horizon strategy as a benchmark for problems where $\{(\hat{R}_t, \hat{D}_t) : t \in \mathcal{T}\}$ are random. The N-period rolling horizon strategy solves an integer multicommodity min-cost network flow problem to make the decisions at time period t. This problem is similar to problem (29), but it "spans" only the time periods $\{t, t+1, \ldots, t+N\}$, as opposed to "spanning" the time periods $\{0, \ldots, T\}$. The first time period t in this problem involves the known realization of $(\hat{R}_t, \hat{D}_t)$ and the future time periods $\{t+1, \ldots, t+N\}$ involve the expected values of $\{(\hat{R}_{t+1}, \hat{D}_{t+1}), \ldots, (\hat{R}_{t+N}, \hat{D}_{t+N})\}$. After solving this problem, we only implement the decisions for time period t and solve a similar problem when making the decisions for time period $t+1$. Figure 9 shows the average objective values obtained in the testing iterations, where the white and the gray bars respectively correspond to ADP and the rolling horizon strategy. The results indicate that ADP performs noticeably better than the rolling horizon strategy.

8. Extensions and Final Remarks

In this chapter, we described a modeling framework for large-scale resource allocation problems, along with a fairly flexible algorithmic framework that can be used to obtain good solutions for them. There are still important questions, some of which have already been addressed by the current research and some of which have not, that remain unanswered in this chapter.

Our modeling framework does not put a restriction on the number of dimensions that we can include in the attribute space. On the other hand, our algorithmic framework uses value function approximations of the form $\bar{V}_t^x(R_t^x) = \sum_{a \in \mathcal{A}} \bar{V}_{ta}^x(R_{ta}^x)$, which implicitly assumes one can enumerate all elements of $\mathcal{A}$. This issue is not as serious as the curse of dimensionality mentioned in Section 3, which is related to the number of possible values that the state vector S_t can take, but it can still be a problem. For example, considering the attribute vector in (2) and assuming that there are 100 locations in the transportation network, 10 possible values for the travel time, 8 possible values for the time on duty, 5 possible values

for the number of days away from home and 10 possible vehicle types, we obtain an attribute space that includes 40,000,000 (=$100 \times 10 \times 8 \times 5 \times 10 \times 100$) attribute vectors. In this case, since problem (13) includes at least $|\mathcal{A}|$ constraints, solving this problem would be difficult. We may use the following strategy to deal with this complication. Although $\mathcal{A}$ may include many elements, the number of available resources is usually small. For example, we have several thousand vehicles in the fleet management setting. In this case, we can solve problem (13) by including only a subset of constraints (4) whose right side satisfies $R_{ta} + \hat{R}_{ta} > 0$. This trick reduces the size of these problems. However, after such a reduction, we are not able to compute ϑ^n_{ta} for all $a \in \mathcal{A}$. This difficulty can be remedied by resorting to aggregation strategies; we can approximate ϑ^n_{ta} in (17) by using $\vartheta^n_{ta'}$ for some other attribute vector a' such that a' is "similar" to a and $R_{ta'} + \hat{R}_{ta'} > 0$.

Throughout this chapter, we assumed that there is a single type of resource and all attribute vectors take values in the same attribute space. As mentioned in Section 2, we can include multiple types of resources in our modeling framework by using multiple attribute spaces, say $\mathcal{A}^1, \ldots, \mathcal{A}^N$, and the attribute vectors for different types of resources take values in different attribute spaces. Unfortunately, it is not clear how we can construct good value function approximations when there are multiple types of resources. Research shows that straightforward separable value function approximations of the form $\bar{V}^x_t(R^x_t) = \sum_{n=1}^{N} \sum_{a \in \mathcal{A}^n} \bar{V}^x_{ta}(R^x_{ta})$ do not perform well.

Another complication that frequently arises is the advance information about the realizations of future random variables. For example, it is common that shippers call in advance for future loads in the fleet management setting. The conventional approach in Markov decision processes to address advance information is to include this information in the state vector. This approach increases the number of dimensions of the state vector and it is not clear how to approximate the value function when the state vector includes such an extra dimension.

We may face other complications depending on the problem setting. To name a few for the fleet management setting, the travel times are often highly variable and using expected values of the travel times does not yield satisfactory results. The load pick up windows are almost always flexible; we have to decide not only which loads to cover but also when to cover these loads. The decision-making structure is often decentralized, in the sense that the decisions for the vehicles located at different locations are made by different dispatchers.

9. Bibliographic Remarks

The approximate dynamic programming framework described in this chapter has its roots in stochastic programming, stochastic approximation and dynamic programming. [18], [11], [16], [3] and [27] provide thorough introductions to stochastic programming and stochastic approximation. [25] covers the classical dynamic programming theory, whereas [2] and [31] cover the approximate dynamic programming methods that are more akin to the approach followed in this chapter.

The modeling framework in Section 2 is a simplified version of the one described in [23]. [28] develops a software architecture that maps this modeling framework to software objects. [24] uses this modeling framework for a driver scheduling problem.

The approximate dynamic programming framework in Section 3 captures the essence of a long line of research documented in [20], [21], [13], [14], [19] and [35]. The idea of using simulated trajectories of the system and updating the value function approximations through stochastic approximation-based methods bears close resemblance to temporal differences and Q-learning, which are treated in detail in [30], [41] and [38]. Numerous methods have been proposed to choose a good set of values for the adjustable parameters in the generic value function approximation structure in (15). [2] and [36] propose simulation-based methods, [10] and [1] utilize the linear programming formulation of the dynamic program and [37] uses regression.

[40] and [4] use piecewise-linear functions to construct bounds on the value functions arising from multi-stage stochastic programs, whereas [7] and [6] use piecewise-linear functions to construct approximations to the value functions. The approaches used in these papers are static; they consider all possible realizations of the random variables simultaneously rather than using simulated trajectories of the system to iteratively improve the value function approximations.

In Section 4, the idea of using linear value function approximations is based on [21]. [12] proposes a method, called concave adaptive value estimation, to update piecewise-linear value function approximations. This method also uses a "local" update of the form (19). The methods described in Section 4 to update piecewise-linear value function approximations are based on [33], [22] and [17].

Scenario-based stochastic programming methods described in Section 6 date back to [9]. [42] and [43] treat these methods in detail. There are several variants of Benders decomposition-based methods; L-Shaped decomposition method, stochastic decomposition method and cutting plane and partial sampling method are three of these. L-shaped decomposition method is due to [39], stochastic decomposition method is due to [15] and cutting plane and partial sampling method is due to [5]. [26] gives a comprehensive treatment of these methods. Stochastic hybrid approximation procedure is due to [8].

Some of the computational results presented in Section 7 are taken from [35].

There is some research that partially answers the questions posed in Section 8. [24] uses the aggregation idea to solve a large-scale driver scheduling problem. [29] systematically investigates different aggregation strategies. [34] and [32] propose value function approximation strategies that allow decentralized decision-making structures. [32] presents a method to address random travel times.

References

[1] D. Adelman. A price-directed approach to stochastic inventory routing. *Operations Research*, 52(4):499–514, 2004.

[2] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

[3] J. R. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer-Verlag, New York, 1997.

[4] J. R. Birge and S. W. Wallace. A separable piecewise linear upper bound for stochastic linear programs. *SIAM J. Control and Optimization*, 26(3):1–14, 1988.

[5] Z. -L. Chen and W. B. Powell. A convergent cutting-plane and partial-sampling algorithm for multistage linear programs with recourse. *Journal of Optimization Theory and Applications*, 103(3):497–524, 1999.

[6] R. K. Cheung and W. B. Powell. An algorithm for multistage dynamic networks with random arc capacities, with an application to dynamic fleet management. *Operations Research*, 44(6):951–963, 1996.

[7] R. K. -M. Cheung and W. B. Powell. Models and algorithms for distribution problems with uncertain demands. *Transportation Science*, 30(1):43–59, 1996.

[8] R. K. -M. Cheung and W. B. Powell. SHAPE-A stochastic hybrid approximation procedure for two-stage stochastic programs. *Operations Research*, 48(1):73–79, 2000.

[9] G. Dantzig and A. Ferguson. The allocation of aircrafts to routes: An example of linear programming under uncertain demand. *Management Science*, 3:45–73, 1956.

[10] D. P. de Farias and B. Van Roy. The linear programming approach to approximate dynamic programming. *Operations Research*, 51(6):850–865, 2003.

[11] Y. Ermoliev and R. J. -B. Wets, editors. *Numerical Techniques for Stochastic Optimization*. Springer-Verlag, New York, 1988.

[12] G. A. Godfrey and W. B. Powell. An adaptive, distribution-free approximation for the newsvendor problem with censored demands, with applications to inventory and distribution problems. *Management Science*, 47(8):1101–1112, 2001.

[13] G. A. Godfrey and W. B. Powell. An adaptive, dynamic programming algorithm for stochastic resource allocation problems I: Single period travel times. *Transportation Science*, 36(1):21–39, 2002.

[14] G. A. Godfrey and W. B. Powell. An adaptive, dynamic programming algorithm for stochastic resource allocation problems II: Multi-period travel times. *Transportation Science*, 36(1):40–54, 2002.

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

[16] P. Kall and S. W. Wallace. *Stochastic Programming*. John Wiley and Sons, New York, 1994.

[17] S. Kunnumkal and H. Topaloglu. Stochastic approximation algorithms and max-norm "projections". Technical report, Cornell University, School of Operations Research and Industrial Engineering, 2005.

[18] H. J. Kushner and D. S. Clark. *Stochastic Approximation Methods for Constrained and Unconstrained Systems*. Springer-Verlang, Berlin, 1978.

[19] K. Papadaki and W. B. Powell. An adaptive dynamic programming algorithm for a stochastic multiproduct batch dispatch problem. *Naval Research Logistics*, 50(7):742–769, 2003.

[20] W. B. Powell and T. A. Carvalho. Dynamic control of multicommodity fleet management problems. *European Journal of Operations Research*, 98:522–541, 1997.

[21] W. B. Powell and T. A. Carvalho. Dynamic control of logistics queueing network for large-scale fleet management. *Transportation Science*, 32(2):90–109, 1998.

[22] W. B. Powell, A. Ruszczynski, and H. Topaloglu. Learning algorithms for separable approximations of stochastic optimization problems. *Mathematics of Operations Research*, 29(4):814–836, 2004.

[23] W. B. Powell, J. A. Shapiro, and H. P. Simão. A representational paradigm for dynamic resource transformation problems. In C. Coullard, R. Fourer, and J. H. Owens, editors, *Annals of Operations Research*, pages 231–279. J.C. Baltzer AG, 2001.

[24] W. B. Powell, J. A. Shapiro, and H. P. Simão. An adaptive dynamic programming algorithm for the heterogeneous resource allocation problem. *Transportation Science*, 36(2):231–249, 2002.

[25] M. L. Puterman. *Markov Decision Processes*. John Wiley and Sons, Inc., New York, 1994.

[26] A. Ruszczynski. Decomposition methods. In A. Ruszczynski and A. Shapiro, editors, Handbook in Operations Research and Management Science*, Volume on* Stochastic Programming, Amsterdam, 2003. North Holland.

[27] A. Ruszczynski and A. Shapiro, editors. *Handbook in Operations Research and Management Science, Volume on Stochastic Programming*. North Holland, Amsterdam, 2003.

[28] J. A. Shapiro. *A Framework for Representing and Solving Dynamic Resource Transformation Problems*. PhD thesis, Department of Operations Research and Financial Engineering, Princeton University, Princeton, NJ, June 1999.

[29] M. Z. Spivey and W. B. Powell. The dynamic assignment problem. *Transportation Science*, 38(4):399–419, 2004.

[30] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[31] R. S. Sutton and A. G. Barto. *Reinforcement Learning*. The MIT Press, Cambridge, MA, 1998.

[32] H. Topaloglu. A parallelizable dynamic fleet management model with random travel times. *European Journal of Operational Research*, to appear.

[33] H. Topaloglu and W. B. Powell. An algorithm for approximating piecewise linear functions from sample gradients. *Operations Research Letters*, 31:66–76, 2003.

[34] H. Topaloglu and W. B. Powell. A distributed decision making structure for dynamic resource allocation using nonlinear functional approximations. *Operations Research*, 53(2):281–297, 2005.

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

[36] J. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42:674–690, 1997.

[37] J. Tsitsiklis and B. Van Roy. Regression methods for pricing complex American-style options. *IEEE Transactions on Neural Networks*, 12(4):694–703, 2001.

[38] J. N. Tsitsiklis. Asynchronous stochastic approximation and Q-learning. *Machine Learning*, 16:185–202, 1994.

[39] R. Van Slyke and R. Wets. L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal of Applied Mathematics*, 17(4):638–663, 1969.

[40] S. W. Wallace. A piecewise linear upper bound on the network recourse function. *Mathematical Programming*, 38:133–146, 1987.

[41] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

[42] R. Wets. Programming under uncertainty: The equivalent convex program. *SIAM Journal of Applied Mathematics*, 14:89–105, 1966.

[43] Roger J. -B. Wets. Stochastic programs with fixed recourse: The equivalent deterministic problem. *SIAM Review*, 16:309–339, 1974.