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

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

Warren B. Powell
Department of Operations Research and Financial Engineering,
Princeton University, Princeton, New Jersey 08544, USA,
powell@princeton.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 results indicate that the strategies we present yield high-quality solutions. We show how our general approach can be used for tactical decisions, such as capacity planning and pricing. 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.

Classical treatment of the problems of this nature is given by Markov decision theory literature. 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, the so-called 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 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 the transportation setting, 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. Our algorithmic framework decomposes the problem into time-staged subproblems and approximates the impact of the current decisions on the future through value function approximations. In Section 4, we provide some insight into the structure of these subproblems. Section 5 describes a variety of alternatives that one can use to improve on the initial value function approximations. Section 6 describes how our algorithmic framework can be used for tactical decisions, such as capacity planning and pricing. In Section 7, we review other possible approaches for solving resource allocation problems, most of which are motivated by the field of stochastic programming. Section 8 presents some computational experiments. We conclude in Section 9 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, and it is somewhat nonstandard. Nevertheless, 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 develop our modeling framework in three dimensions; the resources that are being managed, the decisions through which we can manage the resources and the evolution of our information about the system. To make the ideas concrete, we give examples from the fleet management setting throughout the chapter. For simplicity, we consider the fleet management setting in its most unadorned from; we assume that each vehicle can serve one load at a time and each load has to be carried all the way to its destination once it is picked up.

2.1. Resources

We consider the problem of managing a set of resources over a finite planning horizon $\mathcal{T} = \{1, \ldots, T\}$. Depending on the application, each time period may represent a day, an hour or a millisecond. Also, different time periods can have different lengths. We mostly consider problems that take place over a finite planning horizon, but some of the ideas can be generalized to problems with infinite planning horizons. We define the following to represent the resources.

$\mathcal{A}=$ Attribute space of the resources. We usually use a to denote a generic element of the attribute space and refer to a as an attribute vector. We use a_i to denote the i-th element of a and refer to each element of the attribute vector a as an attribute.

$r_{at}=$ Number of resources that have attribute vector a at time period t.

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 after a_2 time periods. 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_{at}. This implies that we can "put" the vehicles with the same attribute vector in the same "bucket" and treat them as indistinguishable.

Although we mostly consider the case where the resources are indivisible and r_{at} takes integer values, r_{at} may be allowed to take fractional values in general. For example, r_{at} 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.

Throughout the chapter, by suppressing some of the indices in a variable, we denote a vector composed of the components ranging over the suppressed indices. For example, we have $r_t=\{r_{at}:a\in\mathcal{A}\}$. In this case, the vector r_t captures the state of the resources at time period t. We henceforth refer to r_t as the state vector, although completely defining the state of the system may require additional information.

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_{at} 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, then r_{at} gives the inventory count for product 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_{at} 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 $\mathcal{A} = \{1, \ldots, N\} \times \{X_1, \ldots, X_K\}$, and the first element of the attribute vector is static and the second element is dynamic, then $r_{[i,X_k],t}$ gives the number of resources of type i that are in state X_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 and locations.
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 theory literature. 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 theory literature refers to the set of all possible values that the state vector r_t can take. For example, in the fleet management setting, the number of elements of the attribute space $\mathcal{A} = \mathcal{I} \times \{0, 1, \ldots, \mathcal{T}\} \times \mathcal{V}$ 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.

2.2. Controls

Controls are the means by which we can modify the attributes of the resources. We define the following to represent the controls.

$\mathcal{C} =$ Set of decision types.
$\mathcal{D}^c =$ Set of decisions of type c. We usually use d to denote a generic element of $\mathcal{D}^c$. We denote the set of all decisions by $\mathcal{D}$; that is, we have $\mathcal{D} = \bigcup_{c \in \mathcal{C}} \mathcal{D}^c$.
$x_{adt} =$ Number of resources with attribute vector a that are modified by using decision d at time period t.
$c_{adt} =$ Profit contribution from modifying one resource with attribute vector a by using decision d at time period t.

Although one attribute space may be enough, we usually need multiple decision types even in the simplest applications. For example, we may let $\mathcal{C} = \{E, L\}$ to represent the empty repositioning and loaded movement decisions in the fleet management setting. It is best to view a decision d similar to an attribute vector, in the sense that d is a vector taking values in $\mathcal{D}^c$ for some $c \in \mathcal{C}$ and each element of d carries some information about this decision. For example, we may have $d = [d_1, d_2] = [\text{origin}, \text{destination}]$ for $d \in \mathcal{D}^E$ in the fleet management

setting, whereby d captures the decision to move a vehicle empty from location d_1 to d_2. Similarly, we may have

$$d = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} \text{origin} \\ \text{destination} \\ \text{load type} \end{bmatrix} \tag{3}$$

for $d \in \mathcal{D}^L$, which has a similar interpretation as that of $d \in \mathcal{D}^E$, but we now keep track of the type of the load that is served by the loaded movement.

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

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

$$\sum_{d \in \mathcal{D}} x_{adt} = r_{at} \qquad \text{for all } a \in \mathcal{A}. \tag{4}$$

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. An implicit assumption in (4) is that every resource has to be modified by using a decision. Of course, if we do not want to modify a resource, then we can define a "do nothing" decision and apply this decision on the resource. For example, we need a "hold vehicle" decision in the fleet management setting, which keeps the vehicle at its current location. Similarly, we need to apply a "keep on moving" decision on the vehicles that are in-transit at a particular time period. These "hold vehicle" or "keep on moving" decisions can be included in decision type E.

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

$$\delta_{ad}(a') = \begin{cases} 1 & \text{if applying decision } d \text{ on a resource with attribute vector } a \text{ transforms} \\ & \quad \text{the resource into a resource with attribute vector } a' \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

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

$$r_{a',t+1} = \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} \delta_{ad}(a')\, x_{adt} \qquad \text{for all } a' \in \mathcal{A}. \tag{6}$$

Constraints (4) and (6) are basically the flow balance constraints that frequently appear in the network flow literature. As such, we can view the attribute vector-time period pair (a,t) as a node and the decision variable x_{adt} as an arc that leaves this node and enters the node $(a', t+1)$ that satisfies $\delta_{ad}(a') = 1$.

2.3. Evolution of Information

We use the pair of random vectors (O_t, U_t) to represent the information that arrives into the system at time period t. The first component in (O_t, U_t) is a random vector of the form $O_t = \{O_{at} : a \in \mathcal{A}\}$ and O_{at} is the number of resources with attribute vector a that arrive into the system at time period t from an outside source. For some generic index set $\mathcal{K}$, the second component in (O_t, U_t) is a random vector of the form $U_t = \{U_{kt} : k \in \mathcal{K}\}$ and it represents the right side of certain constraints that limit our decisions at time period t. Throughout the chapter, we use (o_t, u_t) to denote a particular realization of (O_t, U_t).

Specifically, we assume that the decisions at time period t are subject to constraints of the form

$$\sum_{a\in\mathcal{A}}\sum_{d\in\mathcal{D}}\rho_{adt}(k)\,x_{adt}\leq u_{kt} \qquad \text{for all } k\in\mathcal{K}, \tag{7}$$

where $\{\rho_{adt}(k): a\in\mathcal{A},\ d\in\mathcal{D},\ k\in\mathcal{K}\}$ are constants determined by the physics of the particular problem setting. We refer to constraints (7) as the physics constraints. In the fleet management setting, o_t may represent the vehicles that join the fleet at time period t. Letting $\mathcal{L}$ be the set of load types, we may have $\mathcal{K}=\mathcal{I}\times\mathcal{I}\times\mathcal{L}$, in which case u_{kt} may represent the number of loads of a certain type that have to be carried between a certain origin-destination pair at time period t.

We mostly assume that U_{kt} takes integer values. Furthermore, we mostly consider the case where $\mathcal{K}=\mathcal{D}$ and focus on two specialized forms of (7) given by

$$\sum_{a\in\mathcal{A}}x_{adt}\leq u_{dt} \qquad \text{for all } d\in\mathcal{D} \tag{8}$$

$$x_{adt}\leq u_{dt} \qquad \text{for all } a\in\mathcal{A},\ d\in\mathcal{D}. \tag{9}$$

Constraints of the form (8) arise when we want to put a limit on the total number of resources that are modified by decision d at time period t. Furthermore, given any decision d, if it is feasible to apply decision d only on the resources with attribute vector $a(d)$, then we have $x_{adt}=0$ for all $a\in\mathcal{A}\setminus\{a(d)\}$ and constraints (8) reduce to

$$x_{a(d),dt}\leq u_{dt} \qquad \text{for all } d\in\mathcal{D}. \tag{10}$$

Constraints (10) can be written as (9) since we have $x_{adt}=0$ for all $a\in\mathcal{A}\setminus\{a(d)\}$. We focus on the special cases in (8) and (9) mainly because they are usually adequate to capture the physics in a variety of applications. Furthermore, the resource allocation problem can be solved as a sequence of min-cost network flow problems or as a sequence of integer multicommodity min-cost network flow problems under these special cases. Section 4 dwells on this issue in detail.

In the fleet management setting, letting u_{dt} be the number of loads that correspond to the loaded movement decision $d=[d_1,d_2,d_3]$ (see (3)), constraints (8) state that the total number of vehicles that are modified by the loaded movement decision d is bounded by the number of loads of type d_3 that need to be carried from location d_1 to d_2. If each load type can be served by a particular vehicle type and a vehicle located at a particular location cannot serve the loads originating at the other locations, then it is feasible to apply the loaded movement decision $d=[d_1,d_2,d_3]$ only on the vehicles with attribute vector $a(d)=[d_1,0,d_3]$ (see (1)). In this case, constraints (8) can be written as constraints (9) by the discussion above.

We clearly take a limited view of the evolution of information. As the system evolves over time, we may find out that a set of resources are temporarily unavailable or we may learn about new cost parameters or we may tune our forecasts of the future random events. These new sources of information may be included in our modeling framework, but we do not need such generalities in the rest of the chapter.

Closing this section, we bring (4), (6) and (7) together by defining

$$\mathcal{X}_t(r_t,o_t,u_t)=\Big\{(x_t,r_{t+1}):\sum_{d\in\mathcal{D}}x_{adt}=r_{at}+o_{at} \qquad \text{for all } a\in\mathcal{A} \tag{11}$$

$$\sum_{a\in\mathcal{A}}\sum_{d\in\mathcal{D}}\delta_{ad}(a')\,x_{adt}-r_{a',t+1}=0 \qquad \text{for all } a'\in\mathcal{A} \tag{12}$$

$$\sum_{a\in\mathcal{A}}\sum_{d\in\mathcal{D}}\rho_{adt}(k)\,x_{adt}\leq u_{kt} \qquad \text{for all } k\in\mathcal{K} \tag{13}$$

$$x_{adt}\in\mathbb{Z}_+ \qquad \text{for all } a\in\mathcal{A},\ d\in\mathcal{D}\Big\}, \tag{14}$$

where we include the arrivals on the right side of the conservation constraints in (11). Therefore, $(x_t, r_{t+1}) \in \mathcal{X}_t(r_t, o_t, u_t)$ means that the decisions x_t are feasible when the state vector is r_t, the realization of arrivals is o_t and the realization of upper bounds is u_t, and applying the decisions x_t generates the state vector r_{t+1} at the next time period.

3. Optimality Criterion and an Algorithmic Framework for Approximate Dynamic Programming

We are interested in finding a Markovian deterministic policy that maximizes the total expected contribution over the whole planning horizon. A Markovian deterministic policy π can be characterized by a sequence of decision functions $\{X_t^\pi(\cdot,\cdot,\cdot) : t \in \mathcal{T}\}$ such that $X_t^\pi(\cdot,\cdot,\cdot)$ maps the state vector r_t, the realization of arrivals o_t and the realization of upper bounds u_t at time period t to a decision vector x_t. One can also define the state transition functions $\{R_{t+1}^\pi(\cdot,\cdot,\cdot) : t \in \mathcal{T}\}$ of policy π such that $R_{t+1}^\pi(\cdot,\cdot,\cdot)$ maps the state vector, the realization of arrivals and the realization of upper bounds at time period t to a state vector for the next time period. We note that given $X_t^\pi(\cdot,\cdot,\cdot)$, $R_{t+1}^\pi(\cdot,\cdot,\cdot)$ can easily be defined by noting the dynamics constraints in (12). In this case, for a given state vector r_t, a realization of future arrivals $\{o_t, \ldots, o_T\}$ and a realization of future upper bounds $\{u_t, \ldots, u_T\}$ at time period t, the cumulative contribution function for policy π can be written recursively as

$$F_t^\pi\big(r_t, o_t, \ldots, o_T, u_t, \ldots, u_T\big) = c_t \cdot X_t^\pi(r_t, o_t, u_t) + F_{t+1}^\pi\big(R_{t+1}^\pi(r_t, o_t, u_t), o_{t+1}, \ldots, o_T, u_{t+1}, \ldots, u_T\big), \quad (15)$$

with the boundary condition $F_{T+1}^\pi(\cdot,\cdot,\ldots,\cdot,\cdot,\ldots,\cdot) = 0$. By repeated application of (15), it is easy to see that $F_1^\pi\big(r_1, o_1, \ldots, o_T, u_1, \ldots, u_T\big)$ is the total contribution obtained over the whole planning horizon when we use policy π, the initial state vector is r_1, the realization of arrivals is $\{o_t : t \in \mathcal{T}\}$ and the realization of upper bounds is $\{u_t : t \in \mathcal{T}\}$.

3.1. Dynamic Programming Formulation

Given the state vector r_t at time period t, if we assume that (O_t, U_t) is conditionally independent of $\{(O_1, U_1), \ldots, (O_{t-1}, U_{t-1})\}$, then it can be shown that the optimal policy π^* satisfying

$$\pi^* = \operatorname*{argmax}_\pi \ \mathbb{E}\big\{F_1^\pi\big(r_1, O_1, \ldots, O_T, U_1, \ldots, U_T\big) \,|\, r_1\big\}$$

is Markovian deterministic. Since (r_t, o_t, u_t) includes all information that we need to make the decisions at time period t, we can use (r_t, o_t, u_t) as the state of the system and find the optimal policy by computing the value functions through the optimality equation

$$V_t(r_t, o_t, u_t) = \max_{(x_t, r_{t+1}) \in \mathcal{X}_t(r_t, o_t, u_t)} c_t \cdot x_t + \mathbb{E}\big\{V_{t+1}(r_{t+1}, O_{t+1}, U_{t+1}) \,|\, r_t, o_t, u_t\big\},$$

with $V_{T+1}(\cdot,\cdot,\cdot) = 0$. Given r_t, o_t and u_t, the maximization operator deterministically defines what x_t and r_{t+1} should be. Therefore, using the conditional independence assumption stated at the beginning of this section, the optimality equation above can be written as

$$V_t(r_t, o_t, u_t) = \max_{(x_t, r_{t+1}) \in \mathcal{X}_t(r_t, o_t, u_t)} c_t \cdot x_t + \mathbb{E}\big\{V_{t+1}(r_{t+1}, O_{t+1}, U_{t+1}) \,|\, r_{t+1}\big\}. \quad (16)$$

In this case, the decision and state transition functions for the optimal policy π^* become

$$\Big(X_t^{\pi^*}(r_t, o_t, u_t), R_{t+1}^{\pi^*}(r_t, o_t, u_t)\Big) = \operatorname*{argmax}_{(x_t, r_{t+1}) \in \mathcal{X}_t(r_t, o_t, u_t)} c_t \cdot x_t + \mathbb{E}\big\{V_{t+1}(r_{t+1}, O_{t+1}, U_{t+1}) \,|\, r_{t+1}\big\}, \quad (17)$$

where we assume that the contribution vector c_t involves small random perturbations so that the problem above always has a unique optimal solution. Under this assumption, the decision and state transition functions are well-defined.

Due to the well-known curse of dimensionality, computing the value functions $\{V_t(\cdot,\cdot,\cdot) : t \in \mathcal{T}\}$ through (16) is intractable for almost every resource allocation problem of practical significance. Our strategy is to construct tractable approximations of the value function. Constructing approximations of the value functions $\{V_t(\cdot,\cdot,\cdot) : t \in \mathcal{T}\}$ is not useful from the practical viewpoint because if we plug the approximation to $V_{t+1}(\cdot,\cdot,\cdot)$ in the right side of (17), then solving problem (17) still requires computing an expectation that involves the random vector (O_{t+1}, U_{t+1}). To work around this difficulty, we let $V_t(r_t) = \mathbb{E}\{V_t(r_t, O_t, U_t) \,|\, r_t\}$ for all $t \in \mathcal{T}$, in which case the optimality equation in (16) can be written as

$$V_t(r_t) = \mathbb{E}\left\{\max_{(x_t, r_{t+1}) \in \mathcal{X}_t(r_t, O_t, U_t)} c_t \cdot x_t + V_{t+1}(r_{t+1}) \,|\, r_t\right\}, \tag{18}$$

with $V_{T+1}(\cdot) = 0$ and (17) becomes

$$\left(X_t^{\pi^*}(r_t, o_t, u_t), R_{t+1}^{\pi^*}(r_t, o_t, u_t)\right) = \operatorname*{argmax}_{(x_t, r_{t+1}) \in \mathcal{X}_t(r_t, o_t, u_t)} c_t \cdot x_t + V_{t+1}(r_{t+1}). \tag{19}$$

In this case, letting $\{\hat{V}_t(\cdot) : t \in \mathcal{T}\}$ be the approximations to the value functions $\{V_t(\cdot) : t \in \mathcal{T}\}$, we plug the approximation $\hat{V}_{t+1}(\cdot)$ in the right side of (19) to make the decisions at time period t. Thus, each set of value function approximations $\{\hat{V}_t(\cdot) : t \in \mathcal{T}\}$ become synonymous with a (suboptimal) policy π whose decision and state transition functions can be written as

$$\left(X_t^{\pi}(r_t, o_t, u_t), R_{t+1}^{\pi}(r_t, o_t, u_t)\right) = \operatorname*{argmax}_{(x_t, r_{t+1}) \in \mathcal{X}_t(r_t, o_t, u_t)} c_t \cdot x_t + \hat{V}_{t+1}(r_{t+1}). \tag{20}$$

If the value function approximations $\{\hat{V}_t(\cdot) : t \in \mathcal{T}\}$ are "close" to the value functions $\{V_t(\cdot) : t \in \mathcal{T}\}$, then the performance of the policy π characterized by the value function approximations $\{\hat{V}_t(\cdot) : t \in \mathcal{T}\}$ should be close to that of the optimal policy π^*. Throughout the chapter, we refer to problem (20) as the approximate subproblem at time period t.

3.2. First Steps Towards Approximate Dynamic Programming

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 approximate subproblems of the form (20) 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 approximate subproblems of the form (21) for all $t \in \mathcal{T}$ is equivalent to simulating the behavior of the policy characterized by the value function approximations $\{\hat{V}_t^n(\cdot) : t \in \mathcal{T}\}$. In Figure 1, we leave the structure of the value function approximations and the inner 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

$$\hat{V}_t(r_t) = \sum_{i \in \mathcal{P}} \alpha_{it}\, \phi_{it}(r_t), \tag{22}$$

where $\{\alpha_{it} : i \in \mathcal{P}\}$ are adjustable parameters and $\{\phi_{it}(\cdot) : i \in \mathcal{P}\}$ are fixed functions. By adjusting $\{\alpha_{it} : i \in \mathcal{P}\}$, we obtain different value function approximations. On the other hand, we can view $\{\phi_{it}(r_t) : i \in \mathcal{P}\}$ as the "essential features" of the state vector r_t that are needed to capture the total expected contribution obtained over the time periods $\{t, \ldots, T\}$.

FIGURE 1. An algorithmic framework for approximate dynamic programming.

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

Step 2. Initialize the time period by letting $t = 1$. Initialize the state vector r_1^n to reflect the initial state of the resources.

Step 3. Sample a realization of (O_t, U_t), say (o_t^n, u_t^n), and solve the approximate subproblem

$$(x_t^n, r_{t+1}^n) = \underset{(x_t, r_{t+1}) \in \mathcal{X}_t(r_t^n, o_t^n, u_t^n)}{\operatorname{argmax}} c_t \cdot x_t + \hat{V}_{t+1}^n(r_{t+1}). \tag{21}$$

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

Step 5. Use the information obtained by solving the approximate subproblems to update the value function approximations. For the moment, we denote this by

$$\{\hat{V}_t^{n+1}(\cdot) : t \in \mathcal{T}\} = \text{Update}\big(\{\hat{V}_t^n(\cdot) : t \in \mathcal{T}\}, \{r_t^n : t \in \mathcal{T}\}, \{(o_t^n, u_t^n) : t \in \mathcal{T}\}\big),$$

where Update$(\cdot)$ can be viewed as a function that maps the value function approximations, the state vectors, the realizations of arrivals and the realizations of upper bounds at iteration n to the value function approximations at iteration $n+1$.

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

The choice of the functions $\{\phi_{it}(\cdot) : i \in \mathcal{P},\ t \in \mathcal{T}\}$ requires some experimentation and some knowledge of the problem structure. However, for given $\{\phi_{it}(\cdot) : i \in \mathcal{P},\ t \in \mathcal{T}\}$, there exist a variety of methods to set the values of the parameters $\{\alpha_{it} : i \in \mathcal{P},\ t\ \in \mathcal{T}\}$ so that the value function approximation in (22) is a good approximation to the value function $V_t(\cdot)$.

For resource allocation problems, we further specialize the value function approximation structure in (22). In particular, we use separable value function approximations of the form

$$\hat{V}_t(r_t) = \sum_{a \in \mathcal{A}} \hat{V}_{at}(r_{at}), \tag{23}$$

where $\{\hat{V}_{at}(\cdot) : a \in \mathcal{A},\ t \in \mathcal{T}\}$ are one-dimensional functions. We focus on two cases.

1. Linear value function approximations. For these value function approximations, we have $\hat{V}_{at}(r_{at}) = \hat{v}_{at}\, r_{at}$, where $\hat{v}_{at}$ are adjustable parameters.

2. Piecewise-linear value function approximations. These value function approximations assume that $\hat{V}_{at}(\cdot)$ 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 $\hat{V}_{at}(\cdot)$ by a sequence of numbers $\{\hat{v}_{at}(q) : q = 1, \ldots, Q\}$, where $\hat{v}_{at}(q)$ is the slope of $\hat{V}_{at}(\cdot)$ over the interval $(q-1, q)$; that is, we have $\hat{v}_{at}(q) = \hat{V}_{at}(q) - \hat{V}_{at}(q-1)$. Since $\hat{V}_{at}(\cdot)$ is concave, we have $\hat{v}_{at}(1) \geq \hat{v}_{at}(2) \geq \ldots \geq \hat{v}_{at}(Q)$.

Linear value function approximations are clearly special cases of (22). To see that piecewise-linear value function approximations are special cases of (22), we assume that $\hat{V}_{at}(0) = 0$ without loss of generality and use $\mathbf{1}(\cdot)$ to denote the indicator function, in which case we have

$$\hat{V}_{at}(r_{at}) = \sum_{q=1}^{r_{at}} \hat{v}_{at}(q) = \sum_{q=1}^{Q} \hat{v}_{at}(q)\, \mathbf{1}(r_{at} \geq q).$$

Letting $\phi_{aqt}(r_t) = \mathbf{1}(r_{at} \geq q)$, we obtain

$$\hat{V}(r_t) = \sum_{a \in \mathcal{A}} \sum_{q=1}^{Q} \hat{v}_{at}(q)\, \phi_{aqt}(r_t),$$

which has the same form as (22). We focus on linear and piecewise-linear value function approximations because these types of value function approximations ensure that the approximate subproblem (20) can be solved by using standard mathematical programming techniques. We elaborate on this issue in Section 4. Furthermore, these types of value function approximations can be described succinctly only by providing some slope information. This feature will be useful when we update and improve the value function approximations. Section 5 dwells on this issue in detail.

3.3. A Note on the Curse of Dimensionality

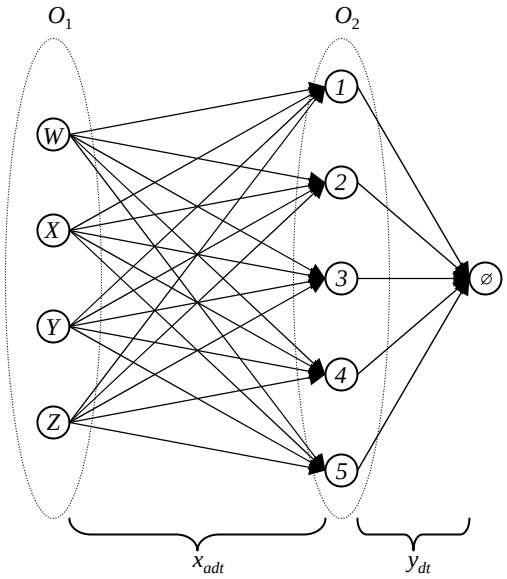
Curse of dimensionality refers to the problematic phenomenon that arises due to the fact that one needs to enumerate all possible values of the state vector r_t to compute the value functions $\{V_t(\cdot) : t \in \mathcal{T}\}$ in (18). This phenomenon is usually brought forward as the main hurdle for the application of dynamic programming techniques on large problems. For resource allocation problems, the large number of possible values for the state vector r_t is clearly a problem, but this is not the only problem. To carry out the computation in (18), we need to compute an expectation that involves the random vector (O_t, U_t). In practice, computing this expectation is almost never tractable. Furthermore, the computation in (18) requires solving the maximization problem in the curly brackets. For resource allocation problems, x_t is usually a high-dimensional vector and enumerating all elements of $\mathcal{X}_t(r_t, O_t, U_t)$ to solve the maximization problem is impossible. Consequently, not only the curse of dimensionality arising from the size of the state space, but also the curses of dimensionality arising from the size of the outcome and the size of the feasible solution space hinder the application of dynamic programming techniques.

The algorithmic framework in Figure 1 stays away from these three curses of dimensionality in the following manner. First, this framework requires simulating the behavior of the policy characterized by the value function approximations $\{\hat{V}_t^n(\cdot) : t \in \mathcal{T}\}$ and it is concerned only with the state vectors $\{r_t^n : t \in \mathcal{T}\}$ generated during the course of the simulation. Enumerating all possible values of the state vector is not required. Second, each simulation works with one set of samples $\{(o_t^n, u_t^n) : t \in \mathcal{T}\}$ and does not require computing an expectation. As we explain in Section 5.2, the Update$(\cdot)$ function essentially "averages" the outcomes of different simulations, but this function is completely tractable. Finally, due to the specific structure of the value function approximations described in Section 3.2, we can solve the approximate subproblem (21) by using standard mathematical programming techniques.

4. Structure of the Approximate Subproblems

In this section, we consider the approximate subproblem (20) under different assumptions for the value function approximations and the physics constraints. For certain cases, we can show that this problem can be solved as a min-cost network flow problem. For others, we need to resort to integer programming techniques. In either case, we emphasize that problem (20) "spans" only one time period and it is much smaller when compared with the full-blown problem that "spans" T time periods. Therefore, resorting to integer programming techniques does not create too much of a nuisance, although we welcome the opportunity to solve problem (20) as a min-cost network flow problem that naturally yields integer solutions.

FIGURE 2. Problem (24) as a min-cost network flow problem.



Note. In this figure and Figures 3 and 4, we assume that $\mathcal{A} = \{W, X, Y, Z\}$ and $\mathcal{D} = \{1, 2, 3, 4, 5\}$.

4.1. Linear Value Function Approximations and Physics Constraints (8)

If the value function approximations are linear functions of the form $V_t(r_t) = \sum_{a \in \mathcal{A}} \hat{v}_{at}\, r_{at}$ and the physics constraints are of the form (8), then the approximate subproblem (20) can be written as

$$\begin{aligned} \max \quad & \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} c_{adt}\, x_{adt} + \sum_{a \in \mathcal{A}} \hat{v}_{a,t+1}\, r_{a,t+1} \\ \text{subject to} \quad & (8), (11), (12), (14). \end{aligned}$$

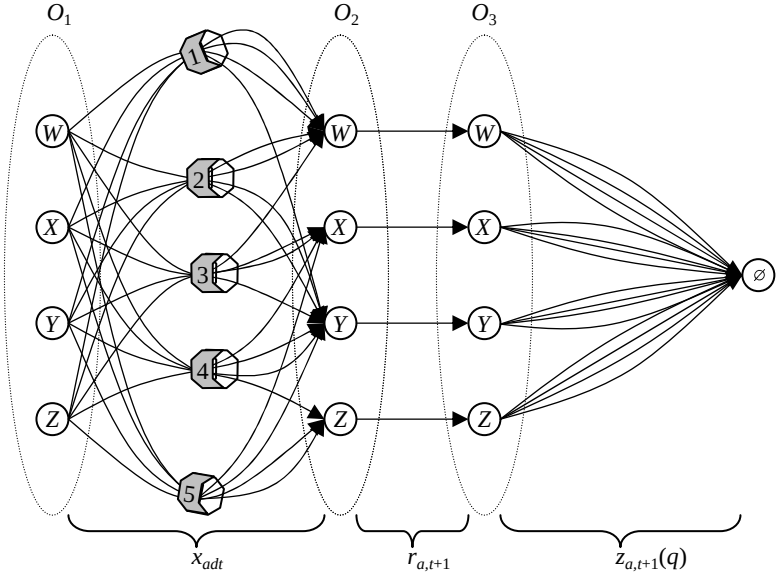
We define new decision variables $\{y_{dt} : d \in \mathcal{D}\}$ and split constraints (8) into $\sum_{a \in \mathcal{A}} x_{adt} - y_{dt} = 0$ and $y_{dt} \leq u_{dt}$ for all $d \in \mathcal{D}$. Using constraints (12) in the objective function, the problem above becomes

$$\begin{aligned} \max \quad & \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} \left\{ c_{adt} + \sum_{a' \in \mathcal{A}} \delta_{ad}(a')\, \hat{v}_{a',t+1} \right\} x_{adt} && && (24) \\ \text{subject to} \quad & \sum_{a \in \mathcal{A}} x_{adt} - y_{dt} = 0 && \text{for all } d \in \mathcal{D} && (25) \\ & y_{dt} \leq u_{dt} && \text{for all } d \in \mathcal{D} \\ & (11), (14). \end{aligned}$$

Defining two sets of nodes $\mathcal{O}_1 = \mathcal{A}$ and $\mathcal{O}_2 = \mathcal{D}$, it is easy to see that problem (24) is a min-cost network flow problem that takes place over a network with the set of nodes $\mathcal{O}_1 \bigcup \mathcal{O}_2 \bigcup \{\emptyset\}$ shown in Figure 2. In this network, there exists an arc corresponding to each decision variable in problem (24). The arc corresponding to decision variable x_{adt} leaves node $a \in \mathcal{O}_1$ and enters node $d \in \mathcal{O}_2$, whereas the arc corresponding to decision variable y_{dt} leaves node $d \in \mathcal{O}_2$ and enters node $\emptyset$. Constraints (11) and (25) in problem (24) are respectively the flow balance constraints for the nodes in $\mathcal{O}_1$ and $\mathcal{O}_2$. The flow balance constraint for node $\emptyset$ is redundant and omitted in problem (24).

Using a similar argument, we can see that the approximate subproblem (20) can be solved as a min-cost network flow problem when we use linear value function approximations and physics constraints (9).

FIGURE 3. Problem (26) as an integer multicommodity min-cost network flow problem.



4.2. Piecewise-Linear Value Function Approximations and Physics Constraints (8)

In this case, assuming that the piecewise-linear function $\hat{V}_{at}(\cdot)$ is characterized by the sequence of slopes $\{\hat{v}_{at}(q) : q = 1, \ldots, Q\}$ as in Section 3.2, the approximate subproblem (20) can be written as

$$\text{max} \quad \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} c_{adt}\, x_{adt} + \sum_{a \in \mathcal{A}} \sum_{q=1}^{Q} \hat{v}_{a,t+1}(q)\, z_{a,t+1}(q) \tag{26}$$

$$\text{subject to} \quad r_{a,t+1} - \sum_{q=1}^{Q} z_{a,t+1}(q) = 0 \qquad \text{for all } a \in \mathcal{A} \tag{27}$$

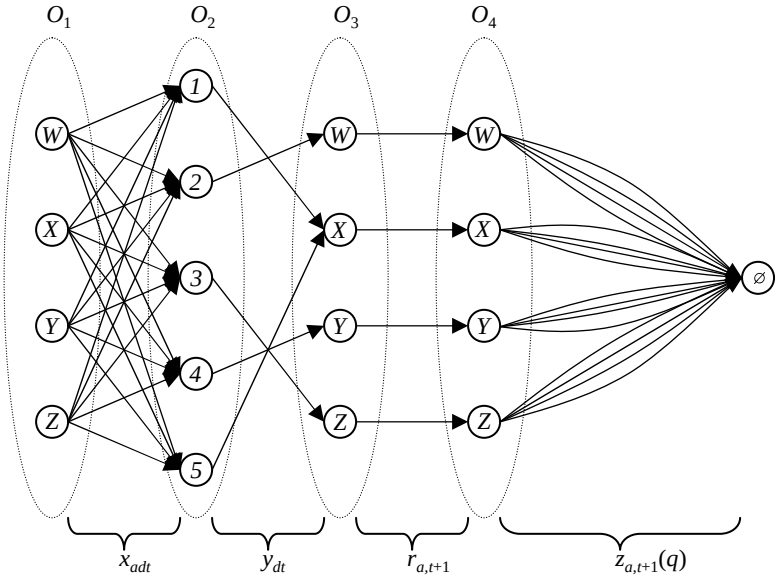
$$z_{a,t+1}(q) \leq 1 \qquad \text{for all } a \in \mathcal{A},\ q = 1, \ldots, Q$$

$$(8), (11), (12), (14),$$

where we use a standard trick to embed the piecewise-linear concave functions $\{\hat{V}_{a,t+1}(\cdot) : a \in \mathcal{A}\}$ into the maximization problem above. Defining three sets of nodes $\mathcal{O}_1 = \mathcal{A}$, $\mathcal{O}_2 = \mathcal{A}$ and $\mathcal{O}_3 = \mathcal{A}$, problem (26) is an integer multicommodity min-cost network flow problem that takes place over a network with the set of nodes $\mathcal{O}_1 \bigcup \mathcal{O}_2 \bigcup \mathcal{O}_3 \bigcup \{\emptyset\}$ shown in Figure 3. The arc corresponding to decision variable x_{adt} leaves node $a \in \mathcal{O}_1$ and enters node $a' \in \mathcal{O}_2$ that satisfies $\delta_{ad}(a') = 1$. The arc corresponding to decision variable $r_{a,t+1}$ leaves node $a \in \mathcal{O}_2$ and enters node $a \in \mathcal{O}_3$. Finally, the arc corresponding to decision variable $z_{a,t+1}(q)$ leaves node $a \in \mathcal{O}_3$ and enters node $\emptyset$. Constraints (11), (12) and (27) in problem (26) are respectively the flow balance constraints for the nodes in $\mathcal{O}_1$, $\mathcal{O}_2$ and $\mathcal{O}_3$. Constraints (8) put a limit on the total flow over subsets of arcs and give problem (26) multicommodity characteristics; the linear programming relaxation of this problem does not necessarily yield integer solutions.

A surprising simplification occurs in problem (26) when the attributes of the resources are "memoryless" in the following sense. For all $a, a' \in \mathcal{A}$, $d \in \mathcal{D}$, assume that value of $\delta_{ad}(a')$ does not depend on a; that is, $\delta_{ad}(a') = \delta_{bd}(a')$ for all $a, a', b \in \mathcal{A}$, $d \in \mathcal{D}$. This is to say that after a resource with attribute vector a is modified by using decision d, its resultant attribute vector only depends on d (see (5)). We now show that problem (26) can be solved as a min-cost network flow problem when the attributes of the resources are "memoryless."

FIGURE 4. Problem (28) as a min-cost network flow problem.



To use the "memorylessness" property, we let $\delta_{\cdot d}(a') = \delta_{ad}(a')$ for all $a, a' \in \mathcal{A}$, $d \in \mathcal{D}$. We define new decision variables $\{y_{dt} : d \in \mathcal{D}\}$ and let $y_{dt} = \sum_{a \in \mathcal{A}} x_{adt}$ for all $d \in \mathcal{D}$. In this case, constraints (12) can be written as

$$\sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} \delta_{ad}(a')\, x_{adt} - r_{a',t+1} = \sum_{d \in \mathcal{D}} \delta_{\cdot d}(a') \sum_{a \in \mathcal{A}} x_{adt} - r_{a',t+1} = \sum_{d \in \mathcal{D}} \delta_{\cdot d}(a')\, y_{dt} - r_{a',t+1} = 0.$$

Thus, problem (26) becomes

$$\begin{aligned}
\max \quad & \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} c_{adt}\, x_{adt} + \sum_{a \in \mathcal{A}} \sum_{q=1}^{Q} \hat{v}_{a,t+1}(q)\, z_{a,t+1}(q) && && (28)\\
\text{subject to} \quad & \sum_{a \in \mathcal{A}} x_{adt} - y_{dt} = 0 && \text{for all } d \in \mathcal{D} && (29)\\
& \sum_{d \in \mathcal{D}} \delta_{\cdot d}(a')\, y_{dt} - r_{a',t+1} = 0 && \text{for all } a' \in \mathcal{A} && (30)\\
& y_{dt} \le u_{dt} && \text{for all } d \in \mathcal{D} &&\\
& r_{a,t+1} - \sum_{q=1}^{Q} z_{a,t+1}(q) = 0 && \text{for all } a \in \mathcal{A} && (31)\\
& z_{a,t+1}(q) \le 1 && \text{for all } a \in \mathcal{A},\ q = 1, \ldots, Q &&\\
& (11), (14). &&
\end{aligned}$$

Defining four sets of nodes $\mathcal{O}_1 = \mathcal{A}$, $\mathcal{O}_2 = \mathcal{D}$, $\mathcal{O}_3 = \mathcal{A}$ and $\mathcal{O}_4 = \mathcal{A}$, this problem is a min-cost network flow problem that takes place over a network with the set of nodes $\mathcal{O}_1 \bigcup \mathcal{O}_2 \bigcup \mathcal{O}_3 \bigcup \mathcal{O}_4 \bigcup \{\emptyset\}$ shown in Figure 4. The arc corresponding to decision variable x_{adt} leaves node $a \in \mathcal{O}_1$ and enters node $d \in \mathcal{O}_2$. The arc corresponding to decision variable y_{dt} leaves node $d \in \mathcal{O}_2$ and enters node $a' \in \mathcal{O}_3$ that satisfies $\delta_{\cdot d}(a') = 1$. The arc corresponding to decision variable $r_{a,t+1}$ leaves node $a \in \mathcal{O}_3$ and enters node $a \in \mathcal{O}_4$. Finally, the arc corresponding to decision variable $z_{a,t+1}(q)$ leaves node $a \in \mathcal{O}_4$ and enters node $\emptyset$. Constraints (11), (29), (30) and (31) are respectively the flow balance constraints for the nodes in $\mathcal{O}_1$, $\mathcal{O}_2$, $\mathcal{O}_3$ and $\mathcal{O}_4$.

In practice, the "memorylessness" property appears in numerous situations. Assuming that we have a single vehicle type in the fleet management setting, the attribute space is

$\mathcal{I} \times \{0, 1, \ldots, \tau\}$. In this case, if we apply the loaded movement decision $d = [d_1, d_2, d_3]$ on a vehicle (see (3)), then the final attribute vector of the vehicle is simply $[d_2, \tau(d_1, d_2)]$, where $\tau(d_1, d_2)$ is the number of time periods required to move from location d_1 to d_2 (see (1)). This final attribute vector depends on d, but not on the initial attribute vector of the vehicle.

The "memorylessness" property does not hold when we have multiple vehicle types and the attribute space is $\mathcal{I} \times \{0, 1, \ldots, \tau\} \times \mathcal{V}$. If we apply the loaded movement decision $d = [d_1, d_2, d_3]$ on a vehicle with attribute vector $a = [d_1, 0, a_3]$ (see (1) and (3)), then the final attribute vector of the vehicle is $[d_2, \tau(d_1, d_2), a_3]$, which depends on the initial attribute vector. One may consider "overloading" the decision with the vehicle type as well; that is, the vehicle type becomes another element in the vector in (3). In this case, if we apply the "overloaded" loaded movement decision $d = [d_1, d_2, d_3, d_4]$ on a vehicle with attribute vector $a = [d_1, 0, d_4]$, then the final attribute vector of the vehicle is $[d_2, \tau(d_1, d_2), d_4]$, which depends only on d and the "memorylessness" property holds. However, the problem here is that when we "overload" the loaded movement decisions in this manner, the physics constraints $\sum_{a \in \mathcal{A}} x_{adt} \leq u_{dt}$ for all $d \in \mathcal{D}$ do not capture the load availability constraints properly. In particular, for an "overloaded" loaded movement decision $d = [d_1, d_2, d_3, d_4]$, u_{dt} represents the number of loads of type d_3 that need to be carried from location d_1 to d_2 by a vehicle of type d_4, whereas we want u_{dt} simply to represent the number of loads of a particular type that need to be carried between a particular origin-destination pair.

The "memorylessness" property may appear in more interesting settings. Consider a chemical trailer that can carry either basic argon gas or purified argon gas. Only "clean" trailers can carry purified argon gas, and once a trailer carries basic argon gas, it becomes "dirty." If the decision "carry purified argon gas" is applied on a trailer, then the trailer remains "clean" after carrying the purified argon gas. On the other hand, if the decision "carry basic argon gas" is applied on a trailer, then the trailer becomes "dirty" irrespective of its status before carrying the basic argon gas. Therefore, the "memorylessness" property holds in this setting.

4.3. Piecewise-Linear Value Function Approximations and Physics Constraints (9)

If the value function approximations are piecewise-linear and the physics constraints are of the form (9), then the approximate subproblem (20) is almost identical to problem (26). The only change we need to make is that we replace constraints (8) in problem (26) with constraints (9). In this case, the approximate subproblem is a min-cost network flow problem that takes place over a network similar to the one in Figure 3. We note that the approximate subproblem is now a min-cost network flow problem because constraints (9) put a limit on the flow over individual arcs and they act as simple upper bound constraints, whereas constraints (8) put a limit on the total flow over subsets of arcs.

5. Constructing the Value Function Approximations

This section describes how we can iteratively update and improve the value function approximations and proposes alternatives for the Update$(\cdot)$ function in the algorithmic framework in Figure 1. In Section 5.1, we consider resource allocation problems whose planning horizons contain only two time periods and present an exact result. Section 5.2 presents approximation methods for more general problems and proposes alternatives for the Update$(\cdot)$ function.

5.1. An Exact Method for Two-Period Problems

We assume that the planning horizon contains two time periods, the physics constraints are of the form (9), and r_2, O_2 and U_2 are independent of each other. When the planning

horizon contains two time periods, we have $V_3(\cdot) = 0$ by definition and all we need is $V_2(\cdot)$, which is used to make the decisions at the first time period.

Under the assumptions stated above, computing $V_2(\cdot)$ is tractable. To see this, omitting the nonnegativity and integrality constraints for brevity, we write (16) as

$$\begin{aligned} V_2(r_2, o_2, u_2) = \max \quad & \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} c_{ad2}\, x_{ad2} & \\ \text{subject to} \quad & \sum_{d \in \mathcal{D}} x_{ad2} = r_{a2} + o_{a2} & \text{for all } a \in \mathcal{A} \\ & x_{ad2} \leq u_{d2} & \text{for all } a \in \mathcal{A},\ d \in \mathcal{D}. \end{aligned}$$

The objective function and constraints above decompose by the elements of $\mathcal{A}$, and letting

$$\begin{aligned} G_{a2}(r_{a2}, u_2) = \max \quad & \sum_{d \in \mathcal{D}} c_{ad2}\, x_{ad2} & \\ \text{subject to} \quad & \sum_{d \in \mathcal{D}} x_{ad2} = r_{a2} & \\ & x_{ad2} \leq u_{d2} & \text{for all } d \in \mathcal{D}, \end{aligned} \tag{32}$$

we have $V_2(r_2, o_2, u_2) = \sum_{a \in \mathcal{A}} G_{a2}(r_{a2} + o_{a2}, u_2)$. We first focus on computing $\hat{G}_{a2}(r_{a2}) = \mathbb{E}\{G_{a2}(r_{a2}, U_2)\}$ for a given value of r_{a2}.

It is easy to see that problem (32) is a knapsack problem, where the items are indexed by $d \in \mathcal{D}$, each item consumes one unit of space, the capacity of the knapsack is r_{a2}, there are u_{d2} available copies of item d and the profit from item d is c_{ad2}. This implies that problem (32) can be solved by a simple sorting operation. In particular, assuming that $\mathcal{D} = \{1, \ldots, |\mathcal{D}|\}$ with $c_{a12} \geq c_{a22} \geq \ldots \geq c_{a|\mathcal{D}|2}$ without loss of generality and letting $\xi_{d2} = u_{12} + \ldots + u_{d2}$ for all $d \in \{1, \ldots, |\mathcal{D}|\}$, we have

$$G_{a2}(r_{a2}, u_2) = \sum_{d=1}^{|\mathcal{D}|} \mathbf{1}(r_{a2} \geq \xi_{d2})\, u_{d2}\, c_{ad2} + \sum_{d=1}^{|\mathcal{D}|} \mathbf{1}(\xi_{d2} > r_{a2} > \xi_{d-1,2})\, [r_{a2} - \xi_{d-1,2}]\, c_{ad2}, \tag{33}$$

where we let $\xi_{02} = 0$ for notational uniformity. The first term on the right side above captures the fact that starting from the most profitable item, we put all available copies of the items into the knapsack as long as there is space. The second term captures the fact that when we cannot fit all available copies of an item, we "top off" the knapsack and stop filling it. Using (33), we obtain

$$G_{a2}(r_{a2}, u_2) - G_{a2}(r_{a2} - 1, u_2) = \sum_{d=1}^{|\mathcal{D}|} \mathbf{1}(\xi_{d2} \geq r_{a2} > \xi_{d-1,2})\, c_{ad2}$$

through elementary algebraic manipulations. Defining the random variable $\gamma_{d2} = U_{12} + \ldots + U_{d2}$ for all $d \in \{1, \ldots, |\mathcal{D}|\}$, taking the expectations of both sides above yields

$$\hat{G}_{a2}(r_{a2}) - \hat{G}_{a2}(r_{a2} - 1) = \sum_{d=1}^{|\mathcal{D}|} \mathbb{P}\{\gamma_{d2} \geq r_{a2} > \gamma_{d-1,2}\}\, c_{ad2}, \tag{34}$$

where we let $\gamma_{02} = 0$ for notational uniformity. The probability on the right side can be simplified as

$$\begin{aligned} \mathbb{P}\{\gamma_{d2} \geq r_{a2} > \gamma_{d-1,2}\} &= \mathbb{P}\{\gamma_{d2} \geq r_{a2}\} + \mathbb{P}\{r_{a2} > \gamma_{d-1,2}\} - \mathbb{P}\{\gamma_{d2} \geq r_{a2} \text{ or } r_{a2} > \gamma_{d-1,2}\} \\ &= \mathbb{P}\{\gamma_{d2} \geq r_{a2}\} + \mathbb{P}\{r_{a2} > \gamma_{d-1,2}\} - 1 + \mathbb{P}\{\gamma_{d2} < r_{a2} \text{ and } r_{a2} \leq \gamma_{d-1,2}\} \\ &= \mathbb{P}\{\gamma_{d2} \geq r_{a2}\} + \mathbb{P}\{r_{a2} > \gamma_{d-1,2}\} - 1. \end{aligned}$$

Thus, noting that $\hat{G}_{a2}(0) = 0$, if we know the probability distribution for the random vector U_t, then we can compute $\hat{G}_{a2}(r_{a2})$ for any value of r_{a2} by using (34) and

$$\hat{G}_{a2}(r_{a2}) = \sum_{q=1}^{r_{a2}} \hat{G}_{a2}(q) - \hat{G}_{a2}(q-1). \tag{35}$$

Finally, to compute the value function, we can use the relationship

$$\begin{aligned}\mathbb{E}\{V_2(r_2, O_2, U_2)\} &= \sum_{a\in\mathcal{A}} \mathbb{E}\{G(r_{a2} + O_{a2}, U_2)\} = \sum_{a\in\mathcal{A}} \mathbb{E}\{\mathbb{E}\{G(r_{a2} + O_{a2}, U_2) \,|\, O_{a2}\}\} \\ &= \sum_{a\in\mathcal{A}} \mathbb{E}\{\hat{G}(r_{a2} + O_{a2})\} = \sum_{a\in\mathcal{A}} \sum_{q=0}^{Q-r_{a2}} \mathbb{P}\{O_{a2} = q\}\, \hat{G}(r_{a2} + q),\end{aligned} \tag{36}$$

where the first and fourth equalities use the independence assumption mentioned at the beginning of this section. Thus, the idea is to compute $\hat{G}_{a2}(r_{a2}) - \hat{G}_{a2}(r_{a2} - 1)$ for all $a \in \mathcal{A}$, $r_{a2} = 1, \ldots, Q$ by using (34), in which case we can compute the value function by using (35) and (36). Since r_{a2} is a scalar, all of these computations are easy.

Letting $V_{a2}(r_{a2}) = \sum_{q=0}^{Q-r_{a2}} \mathbb{P}\{O_{a2} = q\}\, \hat{G}(r_{a2} + q)$, the value function is a separable function of the form $V_2(r_2) = \sum_{a\in\mathcal{A}} V_{a2}(r_{a2})$. Furthermore, it can be shown that $\{V_{a2}(\cdot) : a \in \mathcal{A}\}$ are piecewise-linear concave functions with points of nondifferentiability being a subset of positive integers. Therefore, the results in Section 4.3 apply and the approximate subproblem (20) can be solved as a min-cost network flow problem.

5.2. Monte Carlo Methods for Updating the Value Function Approximations

The method described in Section 5.1 computes the exact value function in (18) under restrictive assumptions. We now go back to the algorithmic framework in Figure 1 that uses approximations of the value functions and iteratively updates these approximations. Our goal is to propose alternatives for the Update$(\cdot)$ function in Step 5 in Figure 1.

We let $\tilde{V}_t^n(r_t^n, o_t^n, u_t^n)$ be the optimal objective value of the approximate subproblem (21); that is, we have

$$\tilde{V}_t^n(r_t^n, o_t^n, u_t^n) = \max_{(x_t, r_{t+1})\in\mathcal{X}_t(r_t^n, o_t^n, u_t^n)} c_t \cdot x_t + \hat{V}_{t+1}^n(r_{t+1}).$$

Whether we use linear or piecewise-linear value function approximations of the form $\hat{V}_t^n(r_t) = \sum_{a\in\mathcal{A}} \hat{V}_{at}^n(r_{at})$, each function $\hat{V}_{at}^n(\cdot)$ 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 $V_t(r_t^n + e_a, o_t^n, u_t^n) - V_t(r_t^n, o_t^n, u_t^n)$ to update and improve the slopes that characterize the function $\hat{V}_{at}^n(\cdot)$. However, this requires knowledge of the exact value function. Instead, we propose using

$$\vartheta_{at}^n(r_t^n, o_t^n, u_t^n) = \tilde{V}_t^n(r_t^n + e_a, o_t^n, u_t^n) - \tilde{V}_t^n(r_t^n, o_t^n, u_t^n). \tag{37}$$

Another advantage of having network flow structure for the approximate subproblems reveals itself in this setting. If the approximate subproblem (21) can be solved as a min-cost network flow problem, then we can compute $\vartheta_{at}^n(r_t^n, o_t^n, u_t^n)$ for all $a \in \mathcal{A}$ through a single min-cost flow augmenting tree computation. For example, the cost of the min-cost flow augmenting path from node $a \in \mathcal{O}_1$ on the left side of Figure 2 to node $\emptyset$ gives $\vartheta_{at}^n(r_t^n, o_t^n, u_t^n)$. (We examine such min-cost flow augmenting paths in more detail in Section 6.) On the other hand, if solving the approximate subproblem (21) requires integer programming techniques, then we can compute $\tilde{V}_t^n(r_t^n + e_a, o_t^n, u_t^n)$ by "physically" perturbing the right side of constraints (11) by

e_a and resolving the problem. For the problem classes that we worked on, the computational effort for perturbing the right side of these constraints and resolving was always manageable because the solution that we obtain when computing $\tilde{V}_t^n(r_t^n, o_t^n, u_t^n)$ usually gives a good starting point when computing $\tilde{V}_t^n(r_t^n + e_a, o_t^n, u_t^n)$.

We first describe 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.

1. Updating linear value function approximations. The method that we use for the linear value function approximations is straightforward. Assuming that the value function approximation at iteration n is of the form $\hat{V}_t^n(r_t) = \sum_{a \in \mathcal{A}} \hat{v}_{at}^n \, r_{at}$, we let

$$\hat{v}_{at}^{n+1} = [1 - \alpha^n] \, \hat{v}_{at}^n + \alpha^n \, \vartheta_{at}^n(r_t^n, o_t^n, u_t^n) \tag{38}$$

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 at iteration $n+1$ is given by $\hat{V}_t^{n+1}(r_t) = \sum_{a \in \mathcal{A}} \hat{v}_{at}^{n+1} \, r_{at}$.

Linear value function approximations are notoriously unstable and experimental work shows that they do not perform as well as piecewise-linear value function approximations. However, our experience indicates that they are better than "nothing;" using linear value function approximations is always better than ignoring the value functions and following a myopic approach. Therefore, linear value function approximations are extremely useful when we deal with a large problem and our only concern is to come up with a strategy that performs better than the myopic approach. Furthermore, it is straightforward to store and update linear value function approximations. As shown in Section 4, linear value function approximations allow us to solve the approximate subproblems as min-cost network flow problems in situations where piecewise-linear value function approximations do not. Finally, linear value function approximations are useful for debugging. Since debugging a model with linear value function approximations is usually much easier, we always try to implement a strategy that uses linear value function approximations as a first step and move on to more sophisticated approximation strategies later.

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

We update $\hat{V}_{at}^n(\cdot)$ by letting

$$\theta_{at}^n(q) = \begin{cases} [1 - \alpha^n] \, \hat{v}_{at}^n(q) + \alpha^n \, \vartheta_{at}^n(r_t^n, o_t^n, u_t^n) & \text{if } q = r_{at}^n + 1 \\ \hat{v}_{at}^n(q) & \text{if } q \in \{1, \ldots, r_{at}^n, r_{at}^n + 2, \ldots, Q\}. \end{cases} \tag{39}$$

The expression above is similar to (38), but the smoothing operation applies only to the "relevant" part of the domain of $\hat{V}_{at}^n(\cdot)$. However, we note that we may not have $\theta_{at}^n(1) \geq \theta_{at}^n(2) \geq \ldots \geq \theta_{at}^n(Q)$, which implies that if we let $\hat{V}_{at}^{n+1}(\cdot)$ be the piecewise-linear function characterized by the sequence of slopes $\{\theta_{at}^n(q) : q = 1, \ldots, Q\}$, then $\hat{V}_{at}^{n+1}(\cdot)$ is not necessarily concave. To make sure that $\hat{V}_{at}^{n+1}(\cdot)$ is concave, we choose a sequence of slopes $\{\hat{v}_{at}^{n+1}(q) : q = 1, \ldots, Q\}$ such that $\{\hat{v}_{at}^{n+1}(q) : q = 1, \ldots, Q\}$ and $\{\theta_{at}^n(q) : q = 1, \ldots, Q\}$ are not too "far" from each other and the sequence of slopes $\{\hat{v}_{at}^{n+1}(q) : q = 1, \ldots, Q\}$ satisfy $\hat{v}_{at}^{n+1}(1) \geq \hat{v}_{at}^{n+1}(2) \geq \ldots \geq \hat{v}_{at}^{n+1}(Q)$. In this case, we let $\hat{V}_{at}^{n+1}(\cdot)$ be the piecewise-linear concave function characterized by the sequence of slopes $\{\hat{v}_{at}^{n+1}(q) : q = 1, \ldots, Q\}$.

There are several methods for choosing the sequence of slopes $\{\hat{v}_{at}^{n+1}(q) : q = 1, \ldots, Q\}$. Letting $\hat{v}_{at}^{n+1} = \{\hat{v}_{at}^{n+1}(q) : q = 1, \ldots, Q\}$, one possible method is

$$\begin{aligned} \hat{v}_{at}^{n+1} = \text{argmin} \quad & \sum_{q=1}^{Q} \left[z_q - \theta_{at}^n(q)\right]^2 \\ \text{subject to} \quad & z_{q-1} - z_q \geq 0 \qquad \text{for all } q = 2, \ldots, Q. \end{aligned} \tag{40}$$

Therefore, this method chooses the vector $\hat{v}_{at}^{n+1}$ as the projection of the vector $\theta_{at}^n = \{\theta_{at}^n(q) : q = 1, \ldots, Q\}$ onto the set $\mathcal{W} = \{z \in \mathbb{R}^Q : z_1 \geq z_2 \geq \ldots \geq z_Q\}$; that is, we have

$$\hat{v}_{at}^{n+1} = \underset{z \in \mathcal{W}}{\text{argmin}} \ \|z - \theta_{at}^n\|_2. \tag{41}$$

Using the Karush-Kuhn-Tucker conditions for problem (40), we can come up with a closed-form expression for the projection in (41). We only state the final result here. Since the vector θ_{at}^n differs from the vector $\hat{v}_{at}^n$ in one component and we have $\hat{v}_{at}^n(1) \geq \hat{v}_{at}^n(2) \geq \ldots \geq \hat{v}_{at}^n(Q)$, there are three possible cases to consider; either $\theta_{at}^n(1) \geq \theta_{at}^n(2) \geq \ldots \geq \theta_{at}^n(Q)$, or $\theta_{at}^n(r_{at}^n) < \theta_{at}^n(r_{at}^n + 1)$, or $\theta_{at}^n(r_{at}^n + 1) < \theta_{at}^n(r_{at}^n + 2)$ should hold. If the first case holds, then we can choose $\hat{v}_{at}^{n+1}$ in (41) as θ_{at}^n and we are done. If the second case holds, then we find the largest $q^* \in \{2, \ldots, r_{at}^n + 1\}$ such that

$$\theta_{at}^n(q^* - 1) \geq \frac{1}{r_{at}^n + 2 - q^*} \sum_{q=q^*}^{r_{at}^n + 1} \theta_{at}^n(q).$$

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

$$\hat{v}_{at}^{n+1}(q) = \begin{cases} \dfrac{1}{r_{at}^n + 2 - q^*} \displaystyle\sum_{q=q^*}^{r_{at}^n + 1} \theta_{at}^n(q) & \text{if } q \in \{q^*, \ldots, r_{at}^n + 1\} \\ \theta_{at}^n(q) & \text{if } q \notin \{q^*, \ldots, r_{at}^n + 1\} \end{cases} \tag{42}$$

satisfies the Karush-Kuhn-Tucker conditions for problem (40). If the third case holds, then one can apply a similar argument. Figure 5.a shows how this method works. The black circles in the top portion of this figure show the sequence of slopes $\{\theta_{at}^n(q) : q = 1, \ldots, Q\}$, whereas the white cirles in the bottom portion show the sequence of slopes $\{\hat{v}_{at}^{n+1}(q) : q = 1, \ldots, Q\}$ computed through (42).

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

$$M^* = \begin{cases} \theta_{at}^n(r_{at} + 1) & \text{if } \theta_{at}^n(1) \geq \theta_{at}^n(2) \geq \ldots \geq \theta_{at}^n(Q) \\ \dfrac{\theta_{at}^n(r_{at}^n) + \theta_{at}^n(r_{at}^n + 1)}{2} & \text{if } \theta_{at}^n(r_{at}^n) < \theta_{at}^n(r_{at}^n + 1) \\ \dfrac{\theta_{at}^n(r_{at}^n + 1) + \theta_{at}^n(r_{at}^n + 2)}{2} & \text{if } \theta_{at}^n(r_{at}^n + 1) < \theta_{at}^n(r_{at}^n + 2), \end{cases} \tag{43}$$

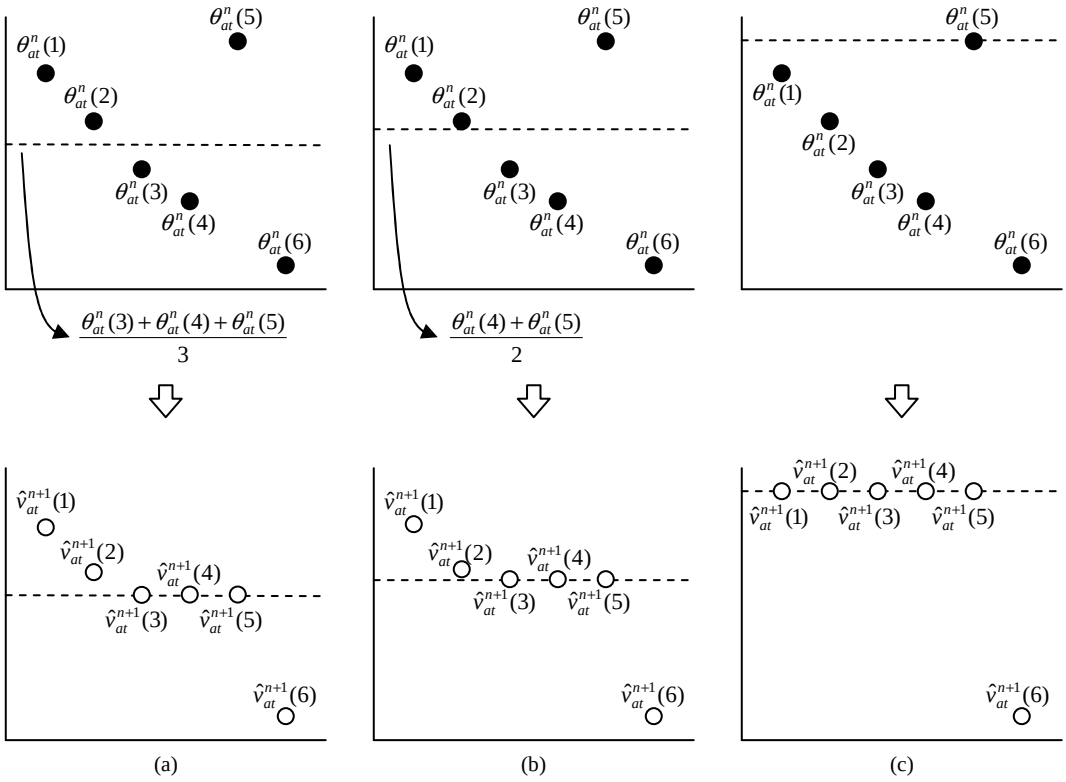
and lets

$$\hat{v}_{at}^{n+1}(q) = \begin{cases} \max\left\{\theta_{at}^n(q),\, M^*\right\} & \text{if } q \in \{1, \ldots, r_{at}^n\} \\ M^* & \text{if } q = r_{at}^n + 1 \\ \min\left\{\theta_{at}^n(q),\, M^*\right\} & \text{if } q \in \{r_{at}^n + 2, \ldots, Q\}. \end{cases} \tag{44}$$

Interestingly, it can be shown that (43) and (44) are equivalent to letting

$$\hat{v}_{at}^{n+1} = \underset{z \in \mathcal{W}}{\text{argmin}} \ \|z - \theta_{at}^n\|_\infty.$$

FIGURE 5. Three possible methods for choosing the vector $\hat{v}_{at}^{n+1}$.



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

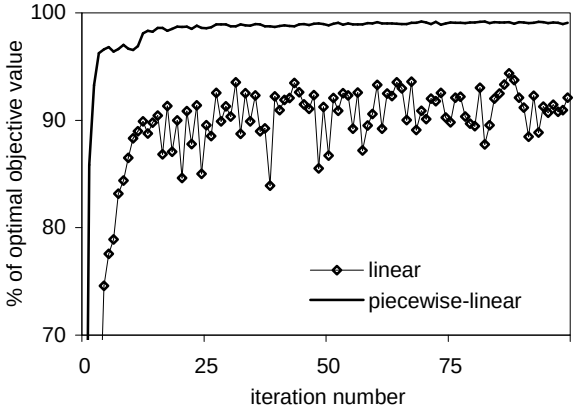
Therefore, the first method is based on a Euclidean-norm projection, whereas the second method is based on a max-norm projection. Figure 5.b shows how this method works.

A slight variation on the second method yields a third method, which computes $M^* = \theta_{at}^n(r_{at}^n+1)$ and lets the vector $\hat{v}_{at}^{n+1}$ be as in (44). This method does not have an interpretation as a projection. Figure 5.c shows how this method works.

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_2^n generated by the algorithmic framework in Figure 1 satisfies $\sum_{n=1}^{\infty}\mathbf{1}(r_2^n=q)=\infty$ 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 $\lim_{n\rightarrow\infty}\hat{v}_2^n(r_2) = V_2(r_2)-V_2(r_2-1)$ for all $r_2=1,\ldots,Q$ with probability 1. (Throughout, we omit the subscript a because the state vector is one-dimensional.) When we apply these methods on 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 provide better objective values and more stable behavior than linear value function approximations. Figure 6 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 6 shows that the policies characterized by piecewise-linear value function approximations may perform almost as well as the optimal solution, whereas

FIGURE 6. Performances of linear and piecewise-linear value function approximations on a resource allocation problem with deterministic data.



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, as mentioned above, 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.

6. Tactical Implications

A question that is often of interest to decision-makers is how different performance measures in a resource allocation model would change in response to changes in certain model parameters. For example, freight carriers are interested in how much their profits would increase if they introduced an additional vehicle into the system or served an additional load. Railroad companies want to estimate the minimum number of railcars necessary to cover the random shipper demands. The Airlift Mobility command is interested in the impact of limited airbase capacities on delayed shipments. Answering such questions requires sensitivity analysis of the underlying model responsible from making the resource allocation decisions.

In this section, we develop sensitivity analysis methods within the context of the algorithmic framework in Figure 1. In particular, we develop sensitivity analysis methods to assess the extra contribution from an additional unit of resource or an additional unit of upper bound introduced into the system. Our approach uses the well-known relationships between the sensitivity analyses of min-cost network flow problems and the min-cost flow augmenting trees. For this reason, among the scenarios considered in Section 4, we focus on the ones under which the approximate subproblem (20) can be solved as a min-cost network flow problem. To make the discussion concrete, we focus on the case where the value function approximations are piecewise-linear and the physics constraints are of the form (10). Under these assumptions, the approximate subproblem can be written as

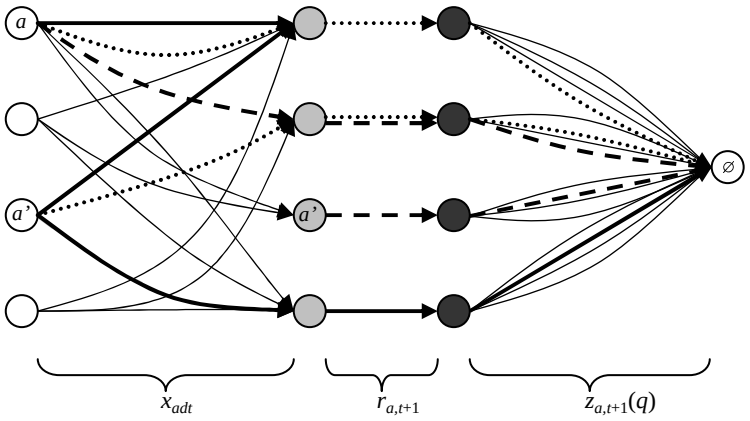
$$\text{max} \quad \sum_{a\in\mathcal{A}}\sum_{d\in\mathcal{D}} c_{adt}\, x_{adt} + \sum_{a\in\mathcal{A}}\sum_{q=1}^{Q} \hat{v}_{a,t+1}(q)\, z_{a,t+1}(q) \tag{45}$$

$$\text{subject to} \quad \sum_{d\in\mathcal{D}} x_{adt} = r_{at} + o_{at} \qquad \text{for all } a\in\mathcal{A} \tag{46}$$

$$\sum_{a\in\mathcal{A}}\sum_{d\in\mathcal{D}} \delta_{ad}(a')\, x_{adt} - r_{a',t+1} = 0 \qquad \text{for all } a'\in\mathcal{A} \tag{47}$$

$$r_{a,t+1} - \sum_{q=1}^{Q} z_{a,t+1}(q) = 0 \qquad \text{for all } a\in\mathcal{A} \tag{48}$$

FIGURE 7. Problem (45) as a min-cost network flow problem.



$$\begin{aligned} &x_{a(d),dt} \le u_{dt} && \text{for all } d \in \mathcal{D} \\ &z_{a,t+1}(q) \le 1 && \text{for all } a \in \mathcal{A},\ q = 1, \ldots, Q \\ &x_{adt},\, r_{a,t+1},\, z_{a,t+1}(q) \in \mathbb{Z}_+ && \text{for all } a \in \mathcal{A},\ d \in \mathcal{D},\ q = 1, \ldots, Q. \end{aligned} \tag{49}$$

Using an argument similar to the one in Section 4, it is easy to see that the problem above is the min-cost network flow problem shown in Figure 7. Constraints (46), (47) and (48) are respectively the flow balance constraints for the white, gray and black nodes. The sets of decision variables $\{x_{adt} : a \in \mathcal{A},\ d \in \mathcal{D}\}$, $\{r_{a,t+1} : a \in \mathcal{A}\}$ and $\{z_{a,t+1}(q) : a \in \mathcal{A},\ q = 1, \ldots, Q\}$ respectively correspond to the arcs that leave the white, gray and black nodes.

6.1. Policy Gradients with Respect to Resource Availabilities

We let π be the policy characterized by the piecewise-linear value function approximations $\{\hat{V}_t(\cdot) : t \in \mathcal{T}\}$, along with the decision and state transition functions defined as in (20). Our first goal is to develop a method to compute how much the total contribution under policy π would change if an additional resource were introduced into the system. We fix a realization of arrivals $o = \{o_t : t \in \mathcal{T}\}$ and a realization of upper bounds $u = \{u_t : t \in \mathcal{T}\}$, and let $\{x_t^{ou} : t \in \mathcal{T}\}$ and $\{r_t^{ou} : t \in \mathcal{T}\}$ be the sequences of decision and state vectors visited by policy π under arrival realizations o and upper bound realizations u; that is, these sequences are recursively computed by

$$x_t^{ou} = X_t^\pi(r_t^{ou}, o_t, u_t) \quad \text{and} \quad r_{t+1}^{ou} = R_{t+1}^\pi(r_t^{ou}, o_t, u_t).$$

Our goal is to develop a tractable method to compute

$$\Phi_t^\pi(e_a, o, u) = F_t^\pi\left(r_t^{ou} + e_a, o_t, \ldots, o_T, u_t, \ldots, u_T\right) - F_t^\pi\left(r_t^{ou}, o_t, \ldots, o_T, u_t, \ldots, u_T\right), \tag{50}$$

where $F_t^\pi(\cdot, \cdot, \ldots, \cdot, \cdot, \ldots, \cdot)$ is as in (15). Therefore, $\Phi_t^\pi(e_a, o, u)$ characterizes how much the total contribution obtained by policy π under arrival realizations o and upper bound realizations u would change if an additional resource with attribute vector a were introduced into the system at time period t.

Constraints (46) are the flow balance constraints for the white nodes on the left side of Figure 7 and the supplies of these nodes are $\{r_{at}^{ou} + o_{at} : a \in \mathcal{A}\}$. Therefore, the vectors

$$\begin{aligned} \xi_t^\pi(e_a, o, u) &= X_t^\pi(r_t^{ou} + e_a, o_t, u_t) - X_t^\pi(r_t^{ou}, o_t, u_t) \\ \Delta_{t+1}^\pi(e_a, o, u) &= R_{t+1}^\pi(r_t^{ou} + e_a, o_t, u_t) - R_{t+1}^\pi(r_t^{ou}, o_t, u_t) \end{aligned}$$

describe how the solution of the min-cost network flow problem in Figure 7 changes in response to a unit increase in the supply of the white node corresponding to the attribute

vector a. It is well-known that $\xi_t^\pi(e_a, o, u)$ and $\Delta_{t+1}^\pi(e_a, o, u)$ can be characterized by a min-cost flow augmenting path from the white node corresponding to the attribute vector a on the left side of Figure 7 to node $\emptyset$. One such flow augmenting path is shown in bold arcs in this figure. Furthermore, since any acyclic path from a white node on the left side of Figure 7 to node $\emptyset$ traverses exactly one of the arcs corresponding to the decision variables $\{r_{a,t+1} : a \in \mathcal{A}\}$, the vector $\Delta_{t+1}^\pi(e_a, o, u)$ is a positive integer unit vector.

In this case, we can use (15) to carry out the computation in (50) as

$$\begin{aligned}\Phi_t^\pi(e_a, o, u) = \; & c_t \cdot \left[X_t^\pi(r_t^{ou} + e_a, o_t, u_t) - X_t^\pi(r_t^{ou}, o_t, u_t)\right] \\ & + F_{t+1}^\pi\big(R_{t+1}^\pi(r_t^{ou} + e_a, o_t, u_t), o_{t+1}, \ldots, o_T, u_{t+1}, \ldots, u_T\big) \\ & - F_{t+1}^\pi\big(R_{t+1}^\pi(r_t^{ou}, o_t, u_t), o_{t+1}, \ldots, o_T, u_{t+1}, \ldots, u_T\big) \\ = \; & c_t \cdot \xi_t^\pi(e_a, o, u) + F_{t+1}^\pi\big(r_{t+1}^{ou} + \Delta_{t+1}^\pi(e_a, o, u), o_{t+1}, \ldots, o_T, u_{t+1}, \ldots, u_T\big) \\ & - F_{t+1}^\pi\big(r_{t+1}^{ou}, o_{t+1}, \ldots, o_T, u_{t+1}, \ldots, u_T\big) \\ = \; & c_t \cdot \xi_t^\pi(e_a, o, u) + \Phi_{t+1}^\pi(\Delta_{t+1}^\pi(e_a, o, u), o, u). \end{aligned} \tag{51}$$

Thus, the idea is to start with the last time period T and let $\Phi_T^\pi(e_a, o, u) = c_T \cdot \xi_T^\pi(e_a, o, u)$ for all $a \in \mathcal{A}$. Since $\Delta_T^\pi(e_a, o, u)$ is always a positive integer unit vector, $\Phi_{T-1}^\pi(e_a, o, u)$ can easily be computed as $c_{T-1} \cdot \xi_{T-1}^\pi(e_a, o, u) + \Phi_T^\pi(\Delta_T^\pi(e_a, o, u), o, u)$. We continue in a similar fashion until we reach the first time period.

6.2. Policy Gradients with Respect to Upper Bounds

In this section, we develop a tractable method to compute

$$\Psi_t^\pi(e'_d, o, u) = F_t^\pi\left(r_t^{ou}, o_t, \ldots, o_T, u_t + e'_d, \ldots, u_T\right) - F_t^\pi\left(r_t^{ou}, o_t, \ldots, o_T, u_t, \ldots, u_T\right), \tag{52}$$

where e'_d denotes the $|\mathcal{D}|$-dimensional unit vector with a 1 in the element corresponding to $d \in \mathcal{D}$. Therefore $\Psi_t^\pi(e'_d, o, u)$ characterizes how much the total contribution obtained by policy π under arrival realizations o and upper bound realizations u would change if an additional unit of upper bound for decision d is made available at time period t.

Since $\{u_{dt} : d \in \mathcal{D}\}$ put a limit on the flow over the arcs that leave the white nodes on the left side of Figure 7, the vectors

$$\begin{aligned}\zeta_t^\pi(e'_d, o, u) &= X_t^\pi(r_t^{ou}, o_t, u_t + e'_d) - X_t^\pi(r_t^{ou}, o_t, u_t) \\ \Theta_{t+1}^\pi(e'_d, o, u) &= R_{t+1}^\pi(r_t^{ou}, o_t, u_t + e'_d) - R_{t+1}^\pi(r_t^{ou}, o_t, u_t)\end{aligned}$$

describe how the solution of the min-cost network flow problem in Figure 7 changes in response to a unit increase in the upper bound of the arc corresponding to the decision variable $x_{a(d),dt}$. Letting $a' \in \mathcal{A}$ be such that $\delta_{a(d),d}(a') = 1$, it is well-known that $\zeta_t^\pi(e'_d, o, u)$ and $\Theta_{t+1}^\pi(e'_d, o, u)$ can be characterized by a min-cost flow augmenting path from the gray node corresponding to the attribute vector a' in the middle section of Figure 7 to the white node corresponding to the attribute vector a on the left side. One such flow augmenting path is shown in dashed arcs in this figure. Furthermore, any acyclic path from a gray node in the middle section of Figure 7 to a white node on the left side traverses either zero or two of the arcs corresponding to the decision variables $\{r_{a,t+1} : a \in \mathcal{A}\}$. Therefore, the vector $\Theta_{t+1}^\pi(e'_d, o, u)$ can be written as

$$\Theta_{t+1}^\pi(e'_d, o, u) = \Theta_{t+1}^{\pi+}(e'_d, o, u) - \Theta_{t+1}^{\pi-}(e'_d, o, u),$$

where each one of the vectors on the right side above is a positive integer unit vector.

In this case, using an argument similar to the one in (51), we obtain

$$\begin{aligned}\Psi_t^\pi(e'_d, o, u) = \; & c_t \cdot \zeta_t^\pi(e'_d, o, u) \\ & + F_{t+1}^\pi\big(r_{t+1}^{ou} + \Theta_{t+1}^{\pi+}(e'_d, o, u) - \Theta_{t+1}^{\pi-}(e'_d, o, u), o_{t+1}, \ldots, o_T, u_{t+1}, \ldots, u_T\big) \\ & - F_{t+1}^\pi\big(r_{t+1}^{ou}, o_{t+1}, \ldots, o_T, u_{t+1}, \ldots, u_T\big). \end{aligned} \tag{53}$$

Finally, we show that we can develop a tractable method to compute the difference of the last two terms on the right side above. Since the vectors $\Theta^{\pi+}_{t+1}(e'_d, o, u)$ and $\Theta^{\pi-}_{t+1}(e'_d, o, u)$ are positive integer unit vectors, this difference is of the form

$$\Pi^{\pi}_t(e_a, -e_{a'}, o, u) = F^{\pi}_t\big(r^{ou}_t + e_a - e_{a'}, o_t, \ldots, o_T, u_t, \ldots, u_T\big) - F^{\pi}_t\big(r^{ou}_t, o_t, \ldots, o_T, u_t, \ldots, u_T\big),$$

which characterizes how much the total contribution obtained by policy π under arrival realizations o and upper bound realizations u would change if a resource with attribute vector a were introduced into and a resource with attribute vector a' were removed from the system at time period t.

The vectors

$$\begin{aligned}\eta^{\pi}_t(e_a, -e_{a'}, o, u) &= X^{\pi}_t(r^{ou}_t + e_a - e_{a'}, o_t, u_t) - X^{\pi}_t(r^{ou}_t, o_t, u_t)\\ \Lambda^{\pi}_{t+1}(e_a, -e_{a'}, o, u) &= R^{\pi}_{t+1}(r^{ou}_t + e_a - e_{a'}, o_t, u_t) - R^{\pi}_{t+1}(r^{ou}_t, o_t, u_t)\end{aligned}$$

describe how the solution of the min-cost network flow problem in Figure 7 changes in response to a unit increase in the supply of the white node corresponding to the attribute vector a and a unit decrease in the supply of the white node corresponding to the attribute vector a'. It is well-known that $\eta^{\pi}_t(e_a, -e_{a'}, o, u)$ and $\Lambda^{\pi}_{t+1}(e_a, -e_{a'}, o, u)$ can be characterized by a min-cost flow augmenting path from the white node corresponding to the attribute vector a on the left side of Figure 7 to the white node corresponding to the attribute vector a'. One such flow augmenting path is show in dotted arcs in this figure. Furthermore, any acyclic path from a white node on the left side of Figure 7 to another white node on the left side traverses either zero or two of the arcs corresponding to the decision variables $\{r_{a,t+1} : a \in \mathcal{A}\}$. Therefore, the vector $\Lambda^{\pi}_{t+1}(e_a, -e_{a'}, o, u)$ can be written as $\Lambda^{\pi}_{t+1}(e_a, -e_{a'}, o, u) = \Lambda^{\pi+}_{t+1}(e_a, -e_{a'}, o, u) - \Lambda^{\pi-}_{t+1}(e_a, -e_{a'}, o, u)$, where each one of the vectors on the right side is a positive integer unit vector. Using an argument similar to the one in (51), we obtain

$$\begin{aligned}\Pi^{\pi}_t(e_a, -e_{a'}, u, o) = {}& c_t \cdot \eta^{\pi}_t(e_a, -e_{a'}, o, u)\\ &+ F^{\pi}_{t+1}\big(r^{ou}_{t+1} + \Lambda^{\pi+}_{t+1}(e_a, -e_{a'}, o, u) - \Lambda^{\pi-}_{t+1}(e_a, -e_{a'}, o, u), o_{t+1}, \ldots, o_T, u_{t+1}, \ldots, u_T\big)\\ &- F^{\pi}_{t+1}\big(r^{ou}_{t+1}, o_{t+1}, \ldots, o_T, u_{t+1}, \ldots, u_T\big)\\ = {}& c_t \cdot \eta^{\pi}_t(e_a, -e_{a'}, o, u) + \Pi^{\pi}_{t+1}\big(\Lambda^{\pi+}_{t+1}(e_a, -e_{a'}, o, u), -\Lambda^{\pi-}_{t+1}(e_a, -e_{a'}, o, u), o, u\big).\end{aligned}$$

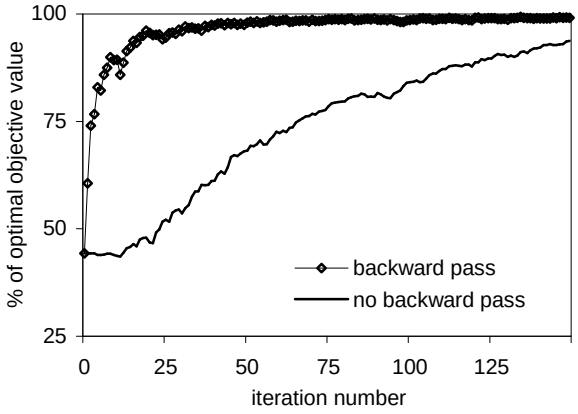
Similar to Section 6.1, the idea is to start with the last time period and let $\Pi^{\pi}_T(e_a, -e_{a'}, o, u) = c_T \cdot \eta^{\pi}_T(e_a, -e_{a'}, o, u)$ for all $a, a' \in \mathcal{A}$. Since $\Lambda^{\pi+}_T(e_a, -e_{a'}, o, u)$ and $\Lambda^{\pi-}_T(e_a, -e_{a'}, o, u)$ are always positive integer unit vectors, $\Pi^{\pi}_{T-1}(e_a, -e_{a'}, o, u)$ can easily be computed as

$$\begin{aligned}\Pi^{\pi}_{T-1}(e_a, -e_{a'}, o, u) = {}& c_{T-1} \cdot \eta^{\pi}_{T-1}(e_a, -e_{a'}, o, u)\\ &+ \Pi^{\pi}_T\big(\Lambda^{\pi+}_T(e_a, -e_{a'}, o, u), -\Lambda^{\pi-}_T(e_a, -e_{a'}, o, u), o, u\big).\end{aligned}$$

We continue in a similar fashion until we reach the first time period. After computing $\{\Pi_t(e_a, -e_{a'}, o, u) : a, a' \in \mathcal{A},\ t \in \mathcal{T}\}$, we can carry out the computation in (53) as $\Psi^{\pi}_t(e'_d, o, u) = c_t \cdot \zeta^{\pi}_t(e'_d, o, u) + \Pi_{t+1}\big(\Theta^{\pi+}_{t+1}(e'_d, o, u), -\Theta^{\pi-}_{t+1}(e'_d, o, u), o, u\big)$.

The obvious use of the methods developed in this section is for tactical decisions. In the fleet management setting, $\Phi^{\pi}_t(e_a, o, u)$ in (50) gives how much the total contribution obtained by policy π would change in response to an additional vehicle with attribute vector a at time period t. On the other hand, $\Psi^{\pi}_t(e'_d, o, u)$ in (52) gives how much the total contribution obtained by policy π would change in response to an additional load that corresponds to the loaded movement decision d at time period t. These quantities can be used for determining the best fleet mix and deciding whether it is profitable to serve a given load. Furthermore, assuming that the load arrivals are price sensitive, we can embed the method described in Section 6.2 in a stochastic gradient-type algorithm to search for the prices that maximize the total expected contribution of policy π.

FIGURE 8. Improvement in the performance through backward pass.



The methods developed in this section can also be used to improve the performance of the algorithmic framework in Figure 1. To illustrate, we let π^n be the policy characterized by the value function approximations $\{\hat{V}_t^n(\cdot) : t \in \mathcal{T}\}$ and $\{r_t^n : t \in \mathcal{T}\}$ be the sequence of state vectors visited by policy π^n under arrival realizations $o^n = \{o_t^n : t \in \mathcal{T}\}$ and upper bound realizations $u^n = \{u_t^n : t \in \mathcal{T}\}$. Roughly speaking, $\mathbb{E}\{\vartheta_{at}^n(r_t^n, O_t, U_t) \,|\, r_t^n\}$ approximates the expected extra contribution from having an additional resource with attribute vector a at time period t (see (37)). However, $\mathbb{E}\{\vartheta_{at}^n(r_t^n, O_t, U_t) \,|\, r_t^n\}$ depends only on the state vector at time period t and the value function approximation at time period $t+1$. Since the value function approximations $\{\hat{V}_t^n(\cdot) : t \in \mathcal{T}\}$ may not represent the expected total contribution of policy π^n very well (especially in the early iterations), $\mathbb{E}\{\vartheta_{at}^n(r_t^n, O_t, U_t) \,|\, r_t^n\}$ may not be a good approximation to the expected extra contribution from an additional resource with attribute vector a at time period t. Although not a huge concern in general, this becomes an issue in applications with extremely long planning horizons or with extremely long travel times. We usually find that using $\Phi_t^{\pi^n}(e_a, o^n, u^n)$ instead of $\vartheta_{at}^n(r_t^n, o_t^n, u_t^n)$ in (38) and (39) significantly improves the performance. When we use $\Phi_t^{\pi^n}(e_a, o^n, u^n)$ in (38) and (39), we say that the Update$(\cdot)$ function uses a backward pass because $\{\Phi_t^{\pi^n}(e_a, o^n, u^n) : a \in \mathcal{A},\ t \in \mathcal{T}\}$ are computed by moving backwards through the time periods. Figure 8 compares the performances of the algorithmic framework in Figure 1 with and without backward pass for a resource allocation problem with deterministic data and indicates that it takes many more iterations to obtain a good policy without backward pass. Usually backward pass is not needed in practice. The problem in Figure 8 is an artificial example from the fleet management setting and involves unrealistically long travel times.

7. Other Approaches for Dynamic Resource Allocation Problems

In this section, we review other alternatives for solving resource allocation problems.

7.1. Formulation as a Deterministic Problem

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

$$\begin{aligned} \max \quad & \sum_{t \in \mathcal{T}} \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} c_{adt}\, x_{adt} && (54) \\ \text{subject to} \quad & \sum_{d \in \mathcal{D}} x_{ad1} = r_{a1} + o_{a1} && \text{for all } a \in \mathcal{A} \end{aligned}$$

$$
\begin{aligned}
& -\sum_{a' \in \mathcal{A}} \sum_{d \in \mathcal{D}} \delta_{a'd}(a)\, x_{a'd,t-1} + \sum_{d \in \mathcal{D}} x_{adt} = \mathbb{E}\{O_{at}\} && \text{for all } a \in \mathcal{A},\ t = 2, \ldots, T \\
& \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} \rho_{ad1}(k)\, x_{ad1} \leq u_{k1} && \text{for all } k \in \mathcal{K} \\
& \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} \rho_{adt}(k)\, x_{adt} \leq \mathbb{E}\{U_{kt}\} && \text{for all } k \in \mathcal{K},\ t = 2, \ldots, T \\
& x_{ad1} \in \mathbb{Z}_+ && \text{for all } a \in \mathcal{A},\ d \in \mathcal{D},
\end{aligned}
$$

where we omit the nonnegativity constraints for time periods $\{2, \ldots, T\}$ for brevity. We do not impose the integrality constraints for time periods $\{2, \ldots, T\}$, because $\{\mathbb{E}\{O_{at}\} : a \in \mathcal{A},\ t = 2, \ldots, T\}$ may be fractional and it may be impossible to satisfy the integrality constraints. The first two sets of constraints are the flow balance constraints, whereas the last two sets of constraints are the physics constraints. For the first time period, which corresponds to here and now, we use the known arrival and upper bound realizations, but we use the expected values of the arrival and upper bound random variables for time periods $\{2, \ldots, T\}$. In practice, we use problem (54) 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 arrival and upper bound realizations at the second time period.

Problem (54) is a large integer program and uses only the expected values of the arrival and upper bound random variables, 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 (54) can efficiently be solved as an integer multicommodity min-cost network flow problem.

7.2. 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 (O_2, U_2), which we denote by $\{(o_2^\omega, u_2^\omega) : \omega \in \Omega\}$. In this case, using p^ω to denote the probability of realization (o_2^ω, u_2^ω), the exact value function at the second time period can be computed by solving

$$
\begin{aligned}
V_2(r_2) = \max \quad & \sum_{\omega \in \Omega} \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} p^\omega\, c_{ad2}\, x_{ad2}^\omega && && (55) \\
\text{subject to} \quad & \sum_{d \in \mathcal{D}} x_{ad2}^\omega = r_{a2} + o_{a2}^\omega && \text{for all } a \in \mathcal{A},\ \omega \in \Omega && (56) \\
& \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} \rho_{ad2}(k)\, x_{ad2}^\omega \leq u_{k2}^\omega && \text{for all } k \in \mathcal{K},\ \omega \in \Omega,
\end{aligned}
$$

where we omit the nonnegativity constraints for brevity throughout this section. This approach allows arbitrary correlation between the random vectors O_2 and U_2, but it assumes that the random vector (O_2, U_2) is independent of r_2. Since the decision variables are $\{x_{ad2}^\omega : a \in \mathcal{A},\ d \in \mathcal{D},\ \omega \in \Omega\}$, problem (55) can be large for practical applications.

7.3. Benders Decomposition-Based Stochastic Programming Methods

Since the state vector r_2 appears on the right side of constraints (56), $V_2(r_2)$ is a piecewise-linear concave function of r_2. Benders decomposition-based methods refer to a class of methods that approximate the exact value function $V_2(\cdot)$ by a series of cuts that are constructed iteratively. In particular, letting $\{\lambda_2^i : i = 1, \ldots, n-1\}$ and $\{\beta_{a2}^i : a \in \mathcal{A},\ i = 1, \ldots, n-1\}$ be the sets of coefficients characterizing the cuts that have been constructed up to iteration n, the function

$$\hat{V}_2^n(r_2) = \min_{i \in \{1,\ldots,n-1\}} \lambda_2^i + \sum_{a \in \mathcal{A}} \beta_{a2}^i \, r_{a2} \tag{57}$$

is the approximation to the exact value function $V_2(\cdot)$ at iteration n. We assume that the cuts that have been constructed up to iteration n provide upper bounds on the exact value function; that is, we have

$$\lambda_2^i + \sum_{a \in \mathcal{A}} \beta_{a2}^i \, r_{a2} \geq V_2(r_2) \qquad \text{for all } i = 1, \ldots, n-1. \tag{58}$$

This implies that $\hat{V}_2^n(r_2) \geq V_2(r_2)$ and the value function approximation in (57) provides an upper bound on the exact value function.

Since we use $\hat{V}_2^n(\cdot)$ as an approximation to $V_2(\cdot)$ at iteration n, the approximate subproblem at the first time period can explicitly be written as

$$\begin{aligned}
\max \quad & \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} c_{ad1} \, x_{ad1} + \hat{v} && && (59)\\
\text{subject to} \quad & \sum_{d \in \mathcal{D}} x_{ad1} = r_{a1} + o_{a1} && \text{for all } a \in \mathcal{A} && \\
& \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} \delta_{ad}(a') \, x_{ad1} - r_{a'2} = 0 && \text{for all } a' \in \mathcal{A} && \\
& \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} \rho_{ad1}(k) \, x_{ad1} \leq u_{k1} && \text{for all } k \in \mathcal{K} && \\
& \hat{v} - \sum_{a \in \mathcal{A}} \beta_{a2}^i \, r_{a2} \leq \lambda_2^i && \text{for all } i = 1, \ldots, n-1 && (60)\\
& \hat{v} \text{ is free.} &&
\end{aligned}$$

We treat the random vector (O_1, U_1) as a constant vector (o_1, u_1), because we solve problem (59) after observing the realizations of the random quantities at the first time period. Letting $(x_1^n, r_2^n, \hat{v}^n)$ be an optimal solution to problem (59), we compute the exact value function for the second time period by solving

$$\begin{aligned}
V_2(r_2^n, o_2^\omega, u_2^\omega) = \max \quad & \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} c_{ad2} \, x_{ad2} && && (61)\\
\text{subject to} \quad & \sum_{d \in \mathcal{D}} x_{ad2} = r_{a2}^n + o_{a2}^\omega && \text{for all } a \in \mathcal{A} && (62)\\
& \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} \rho_{ad2}(k) \, x_{ad2} \leq u_{k2}^\omega && \text{for all } k \in \mathcal{K} && (63)
\end{aligned}$$

for all $\omega \in \Omega$. Using $\{\pi_{a2}^n(\omega) : a \in \mathcal{A}\}$ and $\{\gamma_{k2}^n(\omega) : k \in \mathcal{K}\}$ to denote the optimal values of the dual variables associated with constraints (62) and (63), we have

$$\begin{aligned}
V_2(r_2, o_2^\omega, u_2^\omega) &\leq V_2(r_2^n, o_2^\omega, u_2^\omega) + \sum_{a \in \mathcal{A}} \pi_{a2}^n(\omega) \left[r_{a2} - r_{a2}^n \right] \\
&= \sum_{a \in \mathcal{A}} \pi_{a2}^n(\omega) \left[r_{a2}^n + o_{a2}^\omega \right] + \sum_{k \in \mathcal{K}} \gamma_{k2}^n(\omega) \, u_{k2}^\omega + \sum_{a \in \mathcal{A}} \pi_{a2}^n(\omega) \left[r_{a2} - r_{a2}^n \right].
\end{aligned}$$

Multiplying the expression above by p^ω and adding for all $\omega \in \Omega$, we obtain

$$V_2(r_2) \leq \sum_{\omega \in \Omega} \sum_{a \in \mathcal{A}} p^\omega \, \pi^n_{a2}(\omega) \, o^\omega_{a2} + \sum_{\omega \in \Omega} \sum_{k \in \mathcal{K}} p^\omega \, \gamma^n_{k2}(\omega) \, u^\omega_{k2} + \sum_{\omega \in \Omega} \sum_{a \in \mathcal{A}} p^\omega \, \pi^n_{a2}(\omega) \, r_{a2}. \tag{64}$$

Letting

$$\lambda^n_2 = \sum_{\omega \in \Omega} p^\omega \left\{ \sum_{a \in \mathcal{A}} \pi^n_{a2}(\omega) \, o^\omega_{a2} + \sum_{k \in \mathcal{K}} \gamma^n_{k2}(\omega) \, u^\omega_{k2} \right\} \tag{65}$$

$$\beta^n_{a2} = \sum_{\omega \in \Omega} p^\omega \, \pi^n_{a2}(\omega) \qquad \text{for all } a \in \mathcal{A}, \tag{66}$$

(64) implies that $\lambda^n_2 + \sum_{a \in \mathcal{A}} \beta^n_{a2} \, r_{a2} \geq V_2(r_2)$. Therefore, the cut characterized by λ^n_2 and $\{\beta^n_{a2} : a \in \mathcal{A}\}$ also provides an upper bound on the exact value function (see (58)).

Due to (58) and (60), we always have

$$\hat{v}^n = \min_{i \in \{1, \ldots, n-1\}} \lambda^i_2 + \sum_{a \in \mathcal{A}} \beta^i_{a2} \, r^n_{a2} \geq V_2(r^n_2).$$

After solving problem (61) for all $\omega \in \Omega$ and computing $V_2(r^n_2) = \sum_{\omega \in \Omega} p^\omega \, V_2(r^n_2, o^\omega_2, u^\omega_2)$, if we observe that $\hat{v}^n > V_2(r^n_2)$, then we add the cut $\hat{v} - \sum_{a \in \mathcal{A}} \beta^n_{a2} \, r_{a2} \leq \lambda^n_2$ to the approximate subproblem at the first time period and solve this problem again. Since we have

$$\begin{aligned} \hat{v}^n - \sum_{a \in \mathcal{A}} \beta^n_{a2} \, r^n_{a2} &> V_2(r^n_2) - \sum_{a \in \mathcal{A}} \beta^n_{a2} \, r^n_{a2} \\ &= \sum_{\omega \in \Omega} p^\omega \left\{ \sum_{a \in \mathcal{A}} \pi^n_{a2}(\omega) \left[r^n_{a2} + o^\omega_{a2} \right] + \sum_{k \in \mathcal{K}} \gamma^n_{k2}(\omega) \, u^\omega_{k2} \right\} - \sum_{\omega \in \Omega} \sum_{a \in \mathcal{A}} p^\omega \, \pi^n_{a2}(\omega) \, r^n_{a2} = \lambda^n_2, \end{aligned}$$

the solution $(x^n_1, r^n_2, \hat{v}^n)$ will never be an optimal solution to problem (59) again after adding the cut $\hat{v} - \sum_{a \in \mathcal{A}} \beta^n_{a2} \, r_{a2} \leq \lambda^n_2$. On the other hand, if we observe that $\hat{v} = V_2(r^n_2)$, then we conclude that $(x^n_1, r^n_2, \hat{v}^n)$ gives the optimal decisions at the first time period, because for any other solution $(x_1, r_2) \in \mathcal{X}_1(r_1, o_1, u_1)$, we have

$$\begin{aligned} c_1 \cdot x^n_1 + V_2(r^n_2) = c_1 \cdot x^n_1 + \hat{v}^n &= c_1 \cdot x^n_1 + \min_{i \in \{1, \ldots, n-1\}} \lambda^i_2 + \sum_{a \in \mathcal{A}} \beta^i_{a2} \, r^n_{a2} \\ &\geq c_1 \cdot x_1 + \min_{i \in \{1, \ldots, n-1\}} \lambda^i_2 + \sum_{a \in \mathcal{A}} \beta^i_{a2} \, r_{a2} \geq c_1 \cdot x_1 + V_2(r_2), \end{aligned}$$

where the first inequality uses the fact that $(x^n_1, r^n_2, \hat{v}^n)$ is an optimal solution to problem (59) and the second inequality uses (58).

The method described above is known as L-shaped decomposition. Its biggest computational bottleneck is that constructing a cut requires solving problem (61) for all $\omega \in \Omega$. There are more sophisticated Benders decomposition-based methods that construct a cut by solving problem (61) for one $\omega \in \Omega$. One such method is known as stochastic decomposition. Another important advantage of stochastic decomposition is that it does not require the set of possible realizations for the random vector (O_2, U_2) to be finite. Figure 9 describes another Benders decomposition-based method, known as cutting plane and partial sampling method, which also constructs a cut by solving problem (61) for one $\omega \in \Omega$, but requires the set of possible realizations for the random vector (O_2, U_2) to be finite. Cutting plane and partial sampling method is based on the observation that the set of dual extreme points of problem (61) do not depend on ω. If $\mathcal{P}^n$ includes all dual extreme points of problem (61), then Step 4 is equivalent to solving problem (61) for all $\omega \in \Omega$. In general, $\mathcal{P}^n$ includes only a portion of the dual extreme points and $I(\omega)$ in Step 4 indicates which dual extreme point in $\mathcal{P}^n$ yields the best objective value for $\omega \in \Omega$. In Step 5, we construct the cuts using formulae similar to (65) and (66). When applying cutting plane and partial sampling method in practice, we hope to terminate before $\mathcal{P}^n$ gets too large so that Step 4 can be carried out efficiently.

FIGURE 9. Cutting plane and partial sampling method.

Step 1. Initialize the iteration counter by letting $n = 1$. Initialize the set of available dual solutions by letting $\mathcal{P}^0 = \{\}$.

Step 2. Solve problem (59) and let $(x_1^n, r_2^n, \hat{v}^n)$ be an optimal solution.

Step 3. Let $(o_2^{\omega^n}, u_2^{\omega^n})$ be a sample of (O_2, U_2). Solve problem (61) with $\omega = \omega^n$. Let $\{\pi_{a2}^n : a \in \mathcal{A}\}$ and $\{\gamma_{k2}^n : k \in \mathcal{K}\}$ be the optimal values of the dual variables associated with constraints (62) and (63). Add this dual solution to the set of available dual solutions by letting

$$\mathcal{P}^n = \mathcal{P}^{n-1} \bigcup \left\{ \{\pi_{a2}^n : a \in \mathcal{A}\}, \{\gamma_{k2}^n : k \in \mathcal{K}\} \right\}.$$

Step 4. For all $\omega \in \Omega$, solve problem (61) over the set of available dual solutions; that is, let

$$I(\omega) = \operatorname*{argmin}_{i \in \{1,\ldots,n\}} \quad \sum_{a \in \mathcal{A}} \pi_{a2}^i \left[r_{a2}^n + o_{a2}^\omega \right] + \sum_{k \in \mathcal{K}} \gamma_{k2}^i \, u_{k2}^\omega \qquad \text{for all } \omega \in \Omega.$$

Step 5. Add the cut $\hat{v} - \sum_{a \in \mathcal{A}} \beta_{a2}^n \, r_{a2} \leq \lambda_2^n$ to problem (59), where

$$\lambda_2^n = \sum_{\omega \in \Omega} p^\omega \left\{ \sum_{a \in \mathcal{A}} \pi_{a2}^{I(\omega)} \, o_{a2}^\omega + \sum_{k \in \mathcal{K}} \gamma_{k2}^{I(\omega)} \, u_{k2}^\omega \right\}$$
$$\beta_{a2}^n = \sum_{\omega \in \Omega} p^\omega \, \pi_{a2}^{I(\omega)} \qquad \text{for all } a \in \mathcal{A}.$$

Step 6. Check whether

$$\hat{v}^n = \sum_{\omega \in \Omega} p^\omega \left\{ \sum_{a \in \mathcal{A}} \pi_{a2}^{I(\omega)} \left[r_{a2}^n + o_{a2}^\omega \right] + \sum_{k \in \mathcal{K}} \gamma_{k2}^{I(\omega)} \, u_{k2}^\omega \right\}.$$

If so, then $(x_1^n, r_2^n, \hat{v}^n)$ gives the optimal decisions at the first time period. Otherwise, increase n by 1 and go to Step 2.

7.4. Auxiliary Function-Based Stochastic Programming Methods

As a last possible stochastic programming method, we describe stochastic hybrid approximation procedure. This method is similar to the methods described in Section 5.2; it iteratively updates an approximation to the value function by using a formula similar to (38).

Stochastic hybrid approximation procedure uses value function approximations of the form

$$\hat{V}_2^n(r_2) = \hat{W}_2(r_2) + \sum_{a \in \mathcal{A}} \hat{w}_{a2}^n \, r_{a2}, \tag{67}$$

where $\hat{W}_2(\cdot)$ is a fixed function. Therefore, the first component of the value function approximation does not change over the iterations, but the second (linear) component is adjustable. We assume that $\hat{W}_2(\cdot)$ is a differentiable strongly concave function satisfying

$$\hat{W}_2(\hat{r}_2) - \hat{W}_2(r_2) \leq \sum_{a \in \mathcal{A}} \nabla_a \hat{W}_2(r_2) \left[\hat{r}_{a2} - r_{a2} \right] + B \left\| \hat{r}_2 - r_2 \right\|_2,$$

where the vector $\nabla \hat{W}_2(r_2) = \{\nabla_a \hat{W}_2(r_2) : a \in \mathcal{A}\}$ is the gradient of $\hat{W}_2(\cdot)$ evaluated at r_2 and B is a constant.

Using the value function approximation in (67), we first solve the approximate subproblem at the first time period to obtain

$$(x_1^n, r_2^n) = \underset{(x_1,r_2)\in\mathcal{X}_1(r_1,o_1,u_1)}{\operatorname{argmax}} c_1 \cdot x_1 + \hat{V}_2^n(r_2), \tag{68}$$

where we continue treating the random vector (O_1, U_1) as constant. Letting (o_2^n, u_2^n) be a sample of (O_2, U_2), we solve problem (61) with $o_2^\omega = o^n$ and $u_2^\omega = u_2^n$. In this case, using $\{\pi_{a2}^n : a \in \mathcal{A}\}$ to denote the optimal values of the dual variables associated with constraints (62), we let

$$\hat{w}_{a2}^{n+1} = [1 - \alpha^n]\, \hat{w}_{a2}^n + \alpha^n\, [\pi_{a2}^n - \nabla_a \hat{W}_2(r_2^n)],$$

where $\alpha^n \in [0,1]$ is the smoothing constant at iteration n; the value function approximation at iteration $n+1$ is given by $\hat{V}_2^{n+1}(r_2) = \hat{W}_2(r_2) + \sum_{a\in\mathcal{A}} \hat{w}_{a2}^{n+1}\, r_{a2}$. It is possible to show that $\lim_{n\to\infty} c_1 \cdot x_1^n + V_2(r_2^n) = \min_{(x_1,r_2)\in\mathcal{X}_1(r_1,o_1,u_1)} c_1 \cdot x_1 + V_2(r_2)$ with probability 1. Surprisingly, this result does not require any assumptions on the function $\hat{W}_2(\cdot)$ other than differentiability and strong concavity.

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 $\hat{W}_2(\cdot)$ as a separable quadratic function, then the value function approximation is a separable quadratic function at every iteration. Nevertheless, stochastic hybrid approximation procedure has not seen much attention from the perspective of practical implementations. The first reason for this is that $\hat{W}_2(\cdot)$ is a differentiable function and the approximate subproblem in (68) 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 $\hat{W}_2(\cdot)$ and there is no clear guidelines for this choice. We believe that the methods described in Section 5.2 can be used for this purpose. We can use these methods to construct a piecewise-linear value function approximation, fit a strongly concave separable quadratic function to the piecewise-linear value function approximation and use this fitted function for $\hat{W}_2(\cdot)$.

Figure 10 shows the performances of stochastic hybrid approximation procedure, linear value function approximations and piecewise-linear value function approximations on a resource allocation problem with deterministic data. The objective values obtained by stochastic hybrid approximation procedure 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 stochastic hybrid approximation procedure uses prevents the "bang-bang" behavior of linear value function approximations and provides more stable performance.

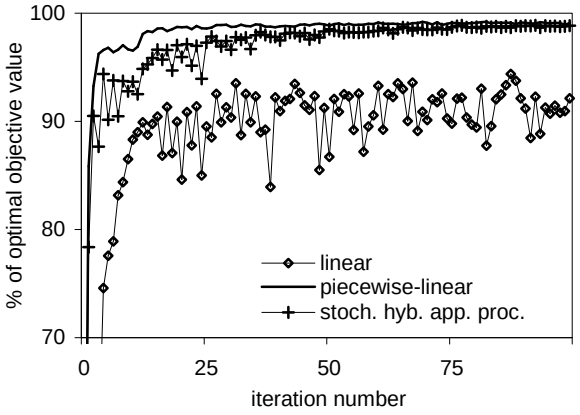
8. Computational Results

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 7, that we can use as benchmarks. This gives us a chance to carefully test the performance of the algorithmic framework in Figure 1.

8.1. 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

FIGURE 10. Performances of stochastic hybrid approximation procedure, and linear and piecewise-linear value function approximations.



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 $\mathcal{C} = \{E, L\}$, where $\mathcal{D}^E$ and $\mathcal{D}^L$ have the same interpretations as in Section 2.2. We assume that there is a single load type and the loaded movement decision in (3) is of the form $d = [\text{origin}, \text{destination}]$. We use piecewise-linear value function approximations and physics constraints of the form (8), in which case since the "memorylessness" property mentioned in Section 4.2 holds, the approximate subproblem (20) can be solved as a min-cost network flow problem. We update the value function approximations by using (39) and (40) 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 random variables $\{U_{dt} : d \in \mathcal{D},\ t \in \mathcal{T}\}$, which represent the numbers of loads between 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 (O_t, U_t) and solve the approximate subproblem (20) 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. 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. Among Benders decomposition-based methods, we use 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.

FIGURE 11. Performances of ADP and CUPPS for different numbers of training iterations.

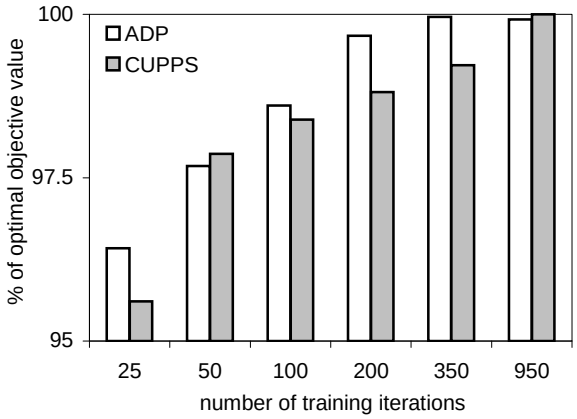
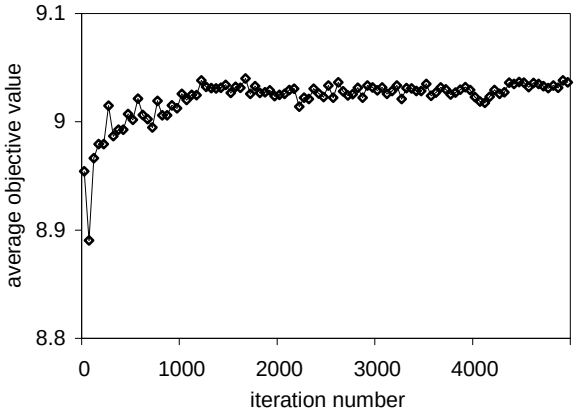


FIGURE 12. Performances of the policies obtained by ADP as a function of the number of training iterations.



For a test problem that involves 30 locations, Figure 11 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 $\{\hat{V}_t^n(\cdot) : t \in \mathcal{T}\}$ be the set of value function approximations obtained by ADP at iteration n, Figure 12 shows the performance of the policy characterized by the value function approximations $\{\hat{V}_t^n(\cdot) : t \in \mathcal{T}\}$ 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 13 shows the average objective values obtained in the testing iterations. In this figure, the number of training 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.

FIGURE 13. Performances of ADP and CUPPS for problems with different numbers of locations.

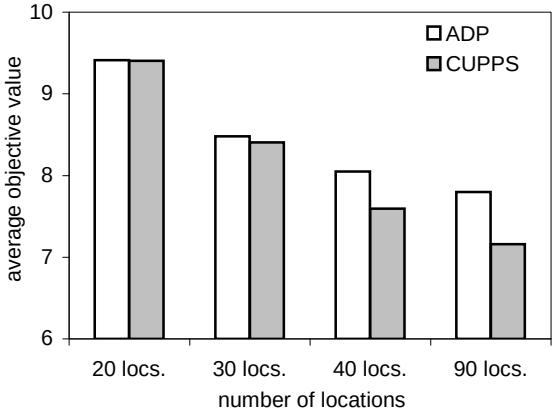


TABLE 1. Performance of ADP on different test problems.

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.

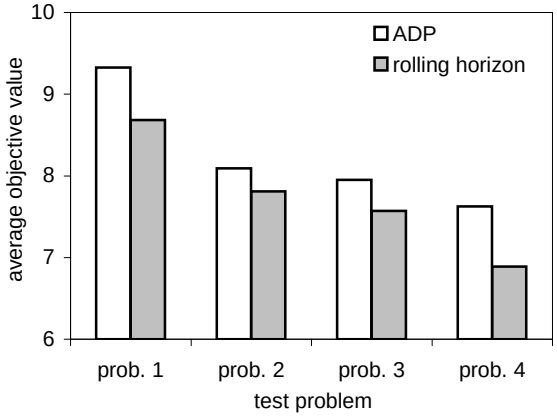
8.2. Multiple-Period Problems

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 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 physics constraints of the form (8), in which case Section 4 shows that the approximate subproblem (20) can be solved as an integer multicommodity min-cost network flow problem. We update the value function approximations by using (39) and (40) with $\alpha^n = 20/(40+n)$.

We begin by exploring the performance of ADP on problems where $\{(O_t,U_t) : t \in \mathcal{T}\}$ are deterministic. These problems can be formulated as integer multicommodity min-cost network flow problems as in problem (54); we solve their linear programming relaxations 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 $\{(O_t,U_t) : t \in \mathcal{T}\}$ 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 (54), but it "spans" only the time periods $\{t, t+1, \ldots, t+N\}$, as opposed to "spanning" the time periods $\{1, \ldots, T\}$. The first time period t in this problem involves the known realization of (O_t,U_t) and the future time periods $\{t+1, \ldots, t+N\}$ involve the expected values of $\{(O_{t+1},U_{t+1}), \ldots, (O_{t+N},U_{t+N})\}$. 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 14 shows the average objective values obtained in the testing iterations. 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.

FIGURE 14. Performances of ADP and the rolling horizon strategy on different test problems.



9. 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 $\hat{V}_t(r_t) = \sum_{a \in \mathcal{A}} \hat{V}_{at}(r_{at})$, 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 3.3, which is related to the number of possible values that the state vector r_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 problems (24), (26) and (28) include at least $|\mathcal{A}|$ constraints, solving them would be problematic. We may use the following strategy to deal with this difficulty. 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 (24), (26) or (28) by including only a subset of constraints (11) whose right side satisfies $r_{at} + o_{at} > 0$. This trick reduces the size of these problems. However, after such a reduction, we are not able to compute $\vartheta^n_{at}(r_t, o_t, u_t)$ for all $a \in \mathcal{A}$. This difficulty can be remedied by resorting to aggregation strategies; we can approximate $\vartheta^n_{at}(r_t, o_t, u_t)$ by using $\vartheta^n_{a't}(r_t, o_t, u_t)$ for some other attribute vector a' such that a' is "similar" to a and $r_{a't} + o_{a't} > 0$.

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 $\hat{V}_t(r_t) = \sum_{n=1}^{N} \sum_{a \in \mathcal{A}^n} \hat{V}_{at}(r_{at})$ 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 theory literature to address advance information is to include this information in the state variable. 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.

10. 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 [29] provide thorough introductions to stochastic programming and stochastic approximation. [27] covers the classical dynamic programming theory, whereas [2] and [33] 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 [25]. [30] develops a software architecture that maps this modeling framework to software objects. [26] 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 [21], [22], [13], [14], [19] and [38]. 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 [32], [45] and [42]. Numerous methods have been proposed to choose a good set of values for the adjustable parameters in the generic value function approximation structure in (22). [2] and [40] propose simulation-based methods, [10] and [1] utilize the linear programming formulation of the dynamic program and [41] uses regression.

[44] 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.

Structures of the approximate subproblems in Section 4 under different value function approximation strategies are investigated in [38].

In Section 5, the method to compute the exact value function for the two-period problem is based on [23], but our derivation using the indicator function is new. [22] uses linear value function approximations and update them by using a method similar to (38). [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 (39). The methods described in Section 5 to update piecewise-linear value function approximations are based on [35], [24] and [17].

The material in Section 6 is taken from [36], but the method to compute $\Psi_t^{\pi}(e'_d, o, u)$ in (52) is new. [39] embeds this method in a stochastic gradient-type algorithm to search for the prices that maximize the total expected contribution of policy π. The well-known relationships between the sensitivity analyses of min-cost network flow problems and the min-cost flow augmenting trees can be found in [20].

Scenario-based stochastic programming methods described in Section 7 date back to [9]. [46] and [47] treat these methods in detail. L-shaped decomposition method is due to [43], but our presentation is closer to that of [28]. Stochastic decomposition, cutting plane and partial sampling method and stochastic hybrid approximation procedure are respectively due to [15], [5] and [8].

Some of the computational results presented in Section 8 are taken from [38].

There is some research that partially answers the questions posed in Section 9. [26] uses the aggregation idea to solve a large-scale driver scheduling problem. [31] systematically investigates different aggregation strategies. [37] and [34] propose value function approximation strategies that allow decentralized decision-making structures. [34] 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. A review of sensitivity results for linear networks and a new approximation to reduce the effects of degeneracy. *Transportation Science*, 23(4):231–243, 1989.

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

[22] 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.

[23] W. B. Powell and R. K. Cheung. Stochastic programs over trees with random arc capacities. *Networks*, 24:161–175, 1994.

[24] 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.

[25] 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.

[26] 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.

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

[28] 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.

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

[30] 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.

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

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

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

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

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

[36] H. Topaloglu and W. B. Powell. Sensitivity analysis of a dynamic fleet management model using approximate dynamic programming. Technical report, Cornell University, School of Operations Research and Industrial Engineering, 2004.

[37] 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.

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

[39] H. Topaloglu and W.B. Powell. Incorporating pricing decisions into the stochastic dynamic fleet management problem. Technical report, Cornell University, School of Operations Research and Industrial Engineering, 2005.

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

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

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

[43] 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.

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

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

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

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