Stochastic Programming in Transportation and Logistics

Warren B. Powell and Huseyin Topaloglu

Department of Operations Research and Financial Engineering, Princeton University, Princeton, NJ 08544

Abstract

Freight transportation is characterized by highly dynamic information processes: customers call in orders over time to move freight; the movement of freight over long distances is subject to random delays; equipment failures require last minute changes; and decisions are not always executed in the field according to plan. The high-dimensionality of the decisions involved has made transportation a natural application for the techniques of mathematical programming, but the challenge of modeling dynamic information processes has limited their success. In this chapter, we explore the use of concepts from stochastic programming in the context of resource allocation problems that arise in freight transportation. Since transportation problems are often quite large, we focus on the degree to which some techniques exploit the natural structure of these problems. Experimental work in the context of these applications is quite limited, so we highlight the techniques that appear to be the most promising.

Preprint submitted to Elsevier Preprint *8 January 2003*

Contents

1 Introduction

Operational models of problems in transportation and logistics offer a ripe set of applications for stochastic programming since they are typically characterized by highly dynamic information processes. In freight transportation, it is the norm to call a carrier the day before, or sometimes the same day, to request that a shipment be moved. In truckload trucking, last minute phone calls are combined with requests that can be made a few days in advance, putting carriers in the position of committing to move loads without knowing the last minute demands that will be made of them (sometimes by their most important customers). In railroads, requests to move freight might be made a week in the future, but it can take a week to move a freight car to a customer. The effect is the same.

The goal of this chapter is to provide some examples of problems, drawn from the arena of freight transportation, that appear to provide a natural application of stochastic programming. Optimization models in transportation and logistics, as they are applied in practice, are almost always formulated based on deterministic models. Our intent is to show where deterministic models can exhibit fundamental weaknesses, not from the perspective of academic theory, but in terms of practical limitations as perceived by people in industry. At the same time, we want to use the richness of real problems to raise issues that may not have been addressed by the stochastic programming community. We want to highlight what works, what does not, and where there are rich areas for new research.

We do not make any effort to provide a comprehensive treatment of stochastic optimization problems in transportation and logistics. First, we consider only problems in freight transportation (for the uninitiated, "transportation and logistics" refers to the operational problems surrounding the movement of goods). These problems are inherently discrete, giving rise to stochastic, integer programming problems, but we focus on problems where linear programming formulations represent a good starting point. We completely avoid the general area of stochastic vehicle routing or the types of batch processes that often arise in the movement of smaller shipments, and focus instead on problems that can be broadly described as dynamic resource allocation problems.

Our presentation begins in section 2 with an overview of different classes of applications. This section provides a summary of the different types of uncertainty that arise, and addresses the fundamental question of why stochastic programming is a promising technology for freight transportation. Section 3 provides a general modeling framework that represents a bridge between linear programming formulations and a representation that more explicitly captures the dimensions of transportation applications. In section 4 we present a case study based on the distribution of freight cars for a railroad. This case study provides us with a problem context where dynamic information processes play an important role. We

use this case study in the remainder of the chapter to keep our discussions grounded in the context of a real application.

We approach the stochastic modeling of our freight car problem in two steps. First, we discuss in section 5 the basic two-stage resource allocation problem. This problem is particularly relevant to the car distribution problem. The characteristics of the car distribution problem nicely illustrate different types of recourse strategies that can arise in practice. Specialized strategies give way to approximations which exploit the underlying network structure. For the most general case (network recourse) we briefly review a broad range of stochastic programming strategies, focusing on their ability to handle the stucture of transportation problems.

Section 6 addresses multistage problems. Our approach toward multistage problems is that they can and should be solved as sequences of two-stage problems. As a result, we solve multistage problems by building on the theory of two-stage problems.

Transportation problems offer far more richness than can be covered in a single chapter. Section 8 provides a hint of the topics that we do not attempt to cover. We close with section 9 that discusses some of the challenges of actually implementing stochastic models in an operational setting.

2 Applications and issues

It is important to have in mind a set of real problems that arise in transportation and logistics. We begin our discussion of applications by listing some sample problems that arise in practice, and then use these problems a) to discuss sources of uncertainty, b) to raise special modeling problems that arise in transportation applications, and finally c) to highlight, from a practical perspective, the limitations of deterministic models and how stochastic programming can improve the quality of our models from a practical perspective.

2.1 Some sample problems

Transportation, fundamentally, is the business of moving things so that they are more useful. If there is a resource at a location i, it may be more useful at another location j. Within this simple framework, there is a tremendous variety of problems that pose special modeling and algorithmic issues. Below is a short list of problems that helps to highlight some of the modeling issues that we will have to grapple with.

1) Product distribution - Perhaps one of the oldest and most practical problems is the determination of how much product to ship from a plant to intermediate warehouses before finally shipping to the retailer (or customer). The decision of how much and where to ship and where must be made before we know the customer demand. There are a number of important variations of this problem, including:
   a) Separability of the distribution process - It is often the case that each customer will be served by a unique warehouse, but substitution among warehouses may be allowed.
   b) Multiple product types with substitution - A company may make multiple product types (for example, different types of salty food snacks) for a market that is willing to purchase different products when one is sold out. For the basic single period distribution problem, substitution between products at different locations is the same as substitution across different types of products, as long as the substitution cost is known (when the cost is a transportation cost, this is known, whereas when it represents the cost of substituting for different product types, it is usually unknown).
   c) Demand backlogging - In multiperiod problems, if demand is not satisfied in one time period, we may assume the demand is lost or backlogged to the next time period. We might add that the same issue arises in the product being managed; highly perishable products vanish if not used at a point in time, whereas nonperishable products stay around.

2) Container management - Often referred to as fleet management in the literature, "containers" represent boxes of various forms that hold freight. These might be trailers, boxcars, or the intermodal containers that are used to move goods across the oceans (and then by truck and rail to inland customers). Containers represent a reusable resource where the act of satisfying a customer demand (moving freight from i to j) also has the effect of changing the state of the system (the container is moved from i and j). The customer demand vanishes from the system, but the container does not. Important problem variations include:
   a) Single commodity problems - These arise when all the containers are the same, or when there are different container types with no substitution between different types of demands. When there is no substitution, the problem decomposes into a series of single commodity problems for each product type.
   b) Multicommodity problems - There may be different container types, and the customers may be willing to substitute between them. For example, they may accept a bigger container, or be willing to move their dry goods in a refrigerated trailer (although no refrigeration is necessary).
   c) Time windows and demand backlogging - The most common model represents customer demands at a point in time, where they are lost if they are not served at that point in time. In practice, it is usually the case that customer orders can be delayed.
   d) Transshipment and relay points - The simplest models represent a demand as the need to move from i to j, and where the movement is represented as a single decision. More

complex operations have to model transportation legs (ocean or rail movements) with relays or transshipment points (ports, rail yards) where the containers move from one mode to the next. A major element of complexity is when capacity constraints are imposed on the transportation legs.

3) Managing complex equipment - The major railroads in North America need to manage fleets of several thousand locomotives. The air mobility command has to move freight and people on a global scale using different types of aircraft. Recently formed companies service a high end market with personal jet service using jets in which the customers own a fraction. These problems have been modeled in the past using the same framework as container management problems with multiple container types. These complex pieces of equipment require something more. For example, there are four major classes of locomotive, reflecting whether they are high or low "adhesion" (a technology that determines the slippage of the wheels on a rail), and whether they are four axle or six axle units (six axle locomotives are more powerful). On closer inspection, we find that the horsepower rating of a locomotive can be divided into 10 or 12 reasonable divisions. It matters if the locomotive has its home shop in Chicago, Atlanta or southern California. Since locomotives may move from the tracks of one railroad to another, it matters who owns the locomotive. And it matters if the locomotive is due into the shop in $1, 2, \ldots, 10$ days, or more than 10 days. In short, complex equipment is complex, and does not lend itself easily to a multicommodity formulation. As we show later, this characteristic determines whether the size of the *attribute space* of a resource is small enough to enumerate the entire space, or too large to enumerate.

4) People and crews - Trucks, trains and planes move because people operate them. Not surprisingly, the modeling of the people is not only important, but requires a set of attributes that makes complex equipment look simple. A truck driver, for example, might be characterized by his current location, his home base, his skill level, whether he has experience driving into Mexico or Canada, how many hours he has driven in the last eight days, how many consecutive hours he has been "on duty" today, and how many hours he has been actively driving during his current duty period. Production systems have to cover these and many other issues.

These problems are all examples of resource allocation problems where, with few exceptions, a single "resource" serves a single "demand." "Bundling" arises when, for example, you need several locomotives to pull a single train, or two drivers (a sleeper team) to operate a single truck. "Layering" arises when you need an aircraft, a pilot, fuel and special loading equipment to move a load from one airbase to another. In some cases, the resource/task dichotomy breaks down. For example, we may be managing locomotives, crews and boxcars. The boxcar needs to go from A to B. We need the crew to move the train, but the crew needs to get back to its home domicile at C. And the locomotive needs

to get to shop at D. We would refer to the locomotives, crew and boxcars as three *resource layers*, since the locomotives, crew and boxcars are all needed to move the train. In fact, for more complex problems, we refer to the objects being managed as *resource layers* (or sometimes, resource classes), where one layer is almost always one that would be referred to as a customer, or job, or task.

2.2 Sources of uncertainty

Uncertainty arises whenever we need to make a decision based on information that is not fully known. We are aware of three scenarios under which this can arise:

1) The information is not yet known, but will become known at some point in the future. This is the standard model of uncertainty.
2) Information is known to someone (or something), but is not known to the decision-maker. We would generally say that this information is *knowable* but for various reasons (most commonly, it is simply too expensive) has not been properly communicated to where the information is needed for a decision.
3) The information will never be known (optimization under incomplete information). For any of a variety of economic or technical reasons, an unknown variable is never measured, even though it would help improve decisions. Since the information is never known, we are not able to develop a probability distribution for it.

Cases (2) and (3) above both represent instances where decisions have to be made without information, but we assume that case (3) represents information that never becomes known explicitly, whereas (2) represents the case where someone knows the information, raising the possibility that the information could be shared (at a cost) or at a minimum, where a probability distribution might be constructed after the fact and shared with others.

Classical uncertainty arises because information arrives over time. It is possible to divide the different types of dynamic information processes into three basic classes: the "resources" being managed (including customer demands), the physical processes that govern the evolution of the system over time, and the decisions that are actually implemented to drive the system. This division reflects our modeling framework, presented in section 3. Since the focus of this volume is on modeling uncertainty, it is useful to give each of these at least a brief discussion.

Resources:
Under the heading of "resources" we include all the information classes that we are actively managing. More formally, these are "endogenously controllable information classes which constrain the system," a definition that includes not just the trucks, trains and planes that

we normally think of as resources, but also the customer orders that these resources are normally serving). Dynamic information processes for resources may include:

a) Information about new (exogenous) arrivals to the system - This normally includes the arrival of customer orders, but may also include the arrivals of the product, equipment or people required to satisfy the customer order. For example, a trucking company is constantly hiring new drivers (there is a lot of turnover) so the arrival of new drivers to the fleet is a dynamic information process. Similarly, a railroad has to manage boxcars, and the process of boxcars becoming empty turns out to be a highly stochastic process (far more uncertain than the customer orders).
b) Information about resources leaving the system - Drivers may quit, locomotives may be retired from service, product can perish. The challenge of modeling departures is that they depend on the state of the system, whereas exogenous arrivals are normally modeled as being independent of the state of the system.
c) Information about the state of a resource - An aircraft may break down or a driver may call in sick.

An important dimension of the modeling of resources is the concept of *knowability* and *actionability*. It is not uncommon for a customer to call in and book an order in advance. Thus, the order becomes known right now (time t) but actionable when it actually arrives to the system at some point in the future (at time $t' \geq t$). Most stochastic models implicitly assume that a customer demand is not known until it actually arrives. By contrast, most deterministic models assume that we know all orders in advance (or more precisely, that we do not want to make a decision taking into account any order that is not already known). In practice, both extremes arise, as well as the case of prebooking where customers call at least some of their orders in advance.

Processes:
Under this category, we include information about parameters that govern the evolution of the system over time. The most important classes include:

a) The time required to complete a decision - In most areas of transportation, travel times are random, and sometimes highly so (although applications vary in the degree to which random arrival times actually matter). In air traffic control problems, planes may land at two minute intervals. Flights of several hours can easily vary in duration by 10 or 20 minutes, so they have to maintain a short backlog of flights to ensure that there is always an aircraft available to land when the runway has the capacity to handle another arrival. In railroads, it is not unusual for the travel time between two points to take anywhere from five to eight days.
b) The cost of a decision - This is often the least uncertain parameter, but there are a number of reasons why we might not know the cost of a decision until after the fact. Costs

which are typically not fully known in advance include tolls, transportation accidents, and processing costs that are not always easy to allocate to a particular activity. Even more uncertain is the revenue that might be received from satisfying a customer which might arise as a result of complex accounting procedures.

c) Parameters that determine the attributes of a resource after a decision - Examples might include the fuel consumption of an aircraft or locomotive (which determines the fuel level), or the maintenance status of the equipment at the end of a trip.

Controls:
In real problems, there is a difference between the decisions that we are planning to make, and the decisions that are actually made. The flow of actual decisions is an important exogenous information process. There are several reasons why an actual physical system does not evolve as planned:

1) The decisions made by a model are not as detailed as what is actually needed in operations. The user has to take a *plan* developed by the model and convert it into something implementable.
2) The user has information not available to the model.
3) The user simply prefers to use a different problem solving approach (possibly suboptimal, but this assumes the solution provided by the model is in some way optimal).

When there is a difference between what a model recommends and the decisions that are actually made, we encounter an instance of the user noncompliance problem. This is a source of uncertainty that is often overlooked.

2.3 Special modeling issues in transportation

Transportation problems introduce an array of issues that provide special modeling and algorithmic challenges. These include:

a) Time staging of information - In freight transportation, information arrives over time. This is the heart of any stochastic model.
b) The lagging of information - Often, a customer will call at time t to place an order to be served at time $t' > t$. The same lagging of information may apply to the vehicles used to serve customers. Since we have information about the future, it is tempting to assume that we can make plans about the future, even before new information becomes known.
c) Complex resource attributes - It is often assumed that the number of different types of resources is "not too large." The number of resource types determines the number of constraints. In practice, the attributes of resources can be surprisingly complex, creating problems where the number of constraints can number in the millions. This is a challenge

even for deterministic models, but poses special difficulties in the context of stochastic problems.

d) Integrality - Many transportation problems exhibit network structure that makes it much easier to obtain integer or near-integer solutions. This structure can be easily destroyed when uncertainty is introduced.

e) Travel times - The common behavior in transportation problems that it takes time to move from one location to the next is generally a minor issue in deterministic models. In stochastic models, it can introduce major complications. If the travel times are deterministic, the result can be a dramatic growth in the size of the state space. However, it is often the case that travel times not only are stochastic, they are not even measurable when the trip is initiated.

f) Multi-agent control - Large transportation systems might be controlled by different agents who control specific dimensions of the system. The decisions of other agents can appear as random variables to a particular agent.

g) Implementation - What we plan may not be the same as what actually happens. An overlooked source of uncertainty is the difference between planned and executed decisions.

2.4 Why do we need stochastic programming?

There are two types of modeling technologies that are widely used in practice: simulation models, which are used almost entirely for planning purposes where there is a need to understand the behavior of a system that evolves over time, and deterministic optimization models and algorithms, when there is a need for the computer to recommend what action should be taken. Stochastic programming brings the modeling of uncertainty explicitly into the process of making a decision (using an optimization algorithm). But, there is a large community of both academic researchers and consultants who feel that they are being quite productive with the algorithms that they are developing based on deterministic models.

There is a broad perception, in both the academic research community and in engineering practice, that deterministic optimization algorithms are "good enough." In part this can be attributed to both the mathematical maturity that has been required to understand stochastic models, and the lack of practical, problem-solving tools. But equally important, we need to understand the ways in which stochastic models can provide solutions that are not just better, but noticeably better in a way that would attract the attention of industry. An understanding of these issues will also indicate where stochastic models are not necessarily appropriate. A partial list of motivations for stochastic models should include:

1) The newsvendor effect - Providing the right amount of resource to meet demand given the uncertainty in demand and the relative costs of providing too much or too little. A deterministic model will never allocate more than the point forecast, even when there are excess resources. Stochastic models can overallocate or underallocate depending on the overall availability of resources to meet forecasted demands.
2) Robust allocation - We might need the container in city A or city C, but we are not sure, so we send the truck halfway in between to city B where it can wait and respond to the demand at the last minute. A deterministic model will never send capacity to a location that does not need it.
3) The value of advance information - Stochastic models can explicit model the staging of information over time. A carrier might want to know the value of having customers book orders farther in advance. A proper analysis of this question needs to consider the value of reducing the uncertainty in a forecast.
4) Forecasts of discrete items - Sometimes it is necessary to forecast low volume demands; for example, orders might be 1 with probability 0.20 and 0 with probability 0.80. A point forecast would produce a demand of 0.20, but a routing and scheduling model is unable to assign 0.20 trucks to the order (the algorithm routes a single truck). Integer rounding amounts to little more than Monte Carlo sampling (simple rounding produces biases - it is necessary to round based on a random sample whose expectation is the same).
5) The algorithmic challenge of solving problems over extended planning horizons - Classical optimization algorithms struggle with optimization problems defined over long horizons, typically as a result of degeneracy. Formulations based on a stochastic "view" of the world produces time-staged problems that are much easier to solve. Sequences of two-stage problems are much easier to solve than a single, large integer program.
6) Overoptimizing problems with imperfect data - A deterministic view of the world can produce problems that are larger and more complex than necessary. An appreciation of uncertainty, not only of the future but also of the "here and now" data (which in practice is a major form of uncertainty) produces models that are smaller and more compact.

3 Modeling framework

The first chapter of this handbook provides a basic mathematical framework for multistage stochastic programming problems. The problem with these abstract formulations is spanning the gap between generic mathematical formulations and real problems. In this section, we offer a notational framework that helps to bridge the gap between real-world dynamic resource allocation problems, and the basic framework of math programming in general, and stochastic programming in particular.

We divide our modeling framework between three fundamental dimensions: the resources being managed, the processes that govern the dynamics of the system, and the structure and organization of controls which manage the system. Our presentation is not the most general, but allows us to focus on the dimensions that are important for modeling the organization and flow of information.

3.1 Resources

To help formalize the discussion, we offer the following definition:

Definition 1 *A resource is an endogenously controllable information class that constrains the system.*

From a math programming perspective, a resource is anything that shows up as a right hand side of a constraint (no surprise that these are often referred to as "resource constraints"). For transportation, resources include trucks, trains, planes, boxcars, containers, drivers/crews, and special equipment that may be needed to complete a trip. Sometimes, but not always, the "demands" being served also meet this definition. For example, the load of freight that we are moving from one location to the next is both endogenously controllable (we often have to determine when to move the load, and sometimes how it is routed) and it constrains the system.

We describe resources using the following:

$\mathcal{C}^R =$ The set of resource classes (e.g. tractors, trailers, drivers, freight).
$\mathcal{R}_c =$ The set of (discrete) resources in class $c \in \mathcal{C}^R$.
$a_r =$ The attributes of resource $r \in \mathcal{R}_c,\ c \in \mathcal{C}^R$.
$\mathcal{A}_c =$ The space of attributes for resource class $c \in \mathcal{C}^R$, with element $a^c \in \mathcal{A}_c$. We often use $\mathcal{A}$ to represent the attribute space of a generic resource.

The attribute vector is a very flexible device for describing the characteristics of a resource. In truckload trucking, it might be the case that all trucks are the same, in which case the attribute vector consists only of the location of the truck. In rail car distribution, the attribute vector can be the type of car as well as the location. If the resource is a human operator, the vector can grow to include attributes such as the home domicile, days away from home, hours of service, and skill sets.

The definition of the attribute space requires an understanding of how a resource evolves over time, and in particular the flow of information. For example, an air cargo carrier working for the military airlift command might have to move a load of cargo from the

eastern United States to southeast Asia. This trip might require midair refueling, as well as stops at several intermediate airbases. Is it necessary to represent the aircraft at each of these intermediate points, or is it enough to assign the aircraft to move a load, and then model its status at the destination? The answer depends on the evolution of information and decisions. For example, if we can completely model all the steps of a trip using the information available when the aircraft first takes off from the origin, then there is no need to model the intermediate points. But we might wish to model the possibility of a failure in the midair refueling, or the failure of the aircraft itself at any of the intermediate airbases. Both of these represent examples of new information arriving to the system, which requires modeling the status of the aircraft just before the new information arrives. The new information may produce new decisions (we may wish to reroute the aircraft) or a change in the dynamics (the aircraft may be unexpectedly delayed at an airbase).

The need to model our aircraft at intermediate points raises a new and even more complex issue. An aircraft that is fully loaded with freight takes on the characteristics of a *layered* (or composite) resource. That is, we have not only the characteristics of the aircraft, but also the characteristics of the freight on the aircraft. This sort of layering arises frequently in transportation operations. Another example arises in the management of locomotives. A locomotive may be sitting idle at a rail yard, or it may be attached to an inbound train (which is making an intermediate stop). If the locomotive is attached to an inbound train, then we have not only the attributes of the locomotive, but also of the train itself (such as its final destination).

We handle this behavior by defining layered attribute vectors. For example, let:

a^A = The attributes of an aircraft.
a^R = The attributes of a load of freight being moved (known as requirements).
a^C = The attributes of the crew piloting the aircraft.

When an aircraft is loaded and making a set of stops, then the attributes of the composite resource at the intermediate stops would be represented using:

$a^{(A)}$ = The attributes of the aircraft *layer*.
= $a^A|a^R|a^C$, where a^A, a^R and a^C are the attributes of the primitive aircraft, requirement and crew resources.

A layer is a concatenation of attributes. An aircraft which is currently sitting idle (a primitive resource) would have the attribute $a^{(A)} = a^A|a^\phi|a^\phi$.

In more complex problems, we may encounter three, four or even five layers. For these problems, we have to define in advance how resources may be combined.

Regardless of our problem class, we let:

$R_{t,a}$ = The number of resources with attribute $a \in \mathcal{A}$ at time t.
$R_t = (R_{t,a})_{a \in \mathcal{A}}$.

One issue that often arises in transportation is the concept of *knowability* and *actionability*. We may *know* of a resource r with attribute a_r at time t which is not *actionable* until some time $t' > t$. This can arise when a customer calls in an order in advance, or when a plane takes off from airport i at time t but will not arrive at airport j until time t'. Actionability can arise as an "estimated time of arrival," an order pickup time, or the time when a task (such as maintenance) will be finished. Actionability can be viewed as being simply an attribute of a resource, and therefore part of the vector a. But often, the actionable time is sufficiently important that it needs to be represented explicitly. In this case, we write:

$R_{t,at'}$ = Number of resources that we know about with attribute a at time t that will not be actionable until time $t' \geq t$.
$R_{tt'} = (R_{t,at'})_{a \in \mathcal{A}}$.
$R_t = (R_{tt'})_{t' \geq t}$.

Thus, we can continue to use the vector R_t as our general state vector, recognizing that it may be divided into elements $R_{tt'}$.

This discussion illustrates a division in the operations research community on the meaning of a time index. Deterministic models of time-staged processes always use time to refer to when an action will happen ("actionability"). Stochastic models almost always use time to refer to the information content of a variable ("knowability" or, in formal terms, "measurability"). In general problems, it is necessary to use both, but this can sometimes be clumsy. We use the double time index (t, t') when we want to explicitly refer to the information content of a variable ("t"), and when an activity actually takes place ("t'"). Whenver we use a single time index, such as R_t, we will always intend the time index to refer to the information content.

3.2 Processes

A dynamic process evolves because of two types of information processes: exogeneous information processes, that arrive as a series of events which update the state of the system, and endogenous information processes, otherwise known as decisions. Following the conventions described in the first chapter of this volume, we let:

ξ_t = The information arriving in time period t. ξ can represent new information about

customer demands, new equipment entering the system, equipment breakdowns, and travel delays.

ξ = $(\xi_t)_{t\in\mathcal{T}}$.

= The information process over the model horizon represented by the set of time periods $\mathcal{T}$.

In general, new information arriving from external sources is captured in a *knowledge base* which summarizes all the information known at time t. Following standard convention, we let $\mathcal{F}_t$ be the $\sigma-$algebra generated by the vector $(\xi_0, \ldots, \xi_t)$.

The standard representation of information in real problems does not always follow standard assumptions. To illustrate, let:

K_t = Our (data) knowledge base at time t.

U^K = The knowledge updating function which updates K_{t-1} using new information ξ_t.

We would representing our updating process as:

$$K_t \leftarrow U^K(K_{t-1}, \xi_t)$$

Realizing that $\mathcal{F}_{t-1} \subseteq \mathcal{F}_t$, one would expect that $\sigma(K_t)$ (the $\sigma-$algebra generated by the random variable K_t) would satisfy $\sigma(K_{t-1}) \subseteq \sigma(K_t)$. This assumes that computer databases do not "forget" information. But this is not always the case. It is not our intent to raise this as a serious issue, but just as a reminder to the reader that standard mathematical assumptions do not always apply to the real world.

For our problems, we can typically divide new information into two classes: the arrivals of new resources (including new customer demands, as well as new equipment or new drivers), and information about model parameters (such as costs and times). This distinction is important in our problem representation, so we define:

$\hat{\rho}_t$ = Updates to model parameters arriving in time period t.

$\hat{R}_{tt'}$ = The vector of new resources arriving in time period t that become actionable at time $t' \geq t$.

$\hat{R}_t$ = $(\hat{R}_{tt'})_{t'\geq t}$.

Thus, we would write $\xi_t = (\hat{\rho}_t, \hat{R}_t)$ with sample realization $\omega_t = \xi_t(\omega) = (\hat{\rho}_t(\omega), \hat{R}_t(\omega))$.

We represent decisions using:

$\mathcal{C}^D$ = The set of decision classes (move empty, move loaded, refuel, maintain the equipment, have a driver go on rest, etc.)

$\mathcal{D}_c =$ The set of discrete decisions in decision class $c \in \mathcal{C}^D$.
$\mathcal{D} = \cup_{c \in \mathcal{C}^D} \mathcal{D}_c$

We use $\mathcal{D}$ to refer to the complete set of decisions. In most transportation applications, it is useful to capture the fact that the set of decisions also depends on the attribute of the resource being acted on. For this purpose we define:

$\mathcal{D}_a =$ The set of decisions that can be used to act on a resource with attribute $a \in \mathcal{A}$.

For the purposes of our presentation, we consider only direct decisions that act on the attributes of a resource (this would exclude, for example, decisions about pricing or what speed to fly an aircraft). For transportation problems, if $d \in \mathcal{D}$ is an instance of a decision, then the impact of the decision is captured through the *modify* function, which is a mapping:

$$M(K_t, a, d) \rightarrow (a', c, \tau) \tag{1}$$

where d is a decision acting on a (possibly layered) resource with attribute a at time t, producing a resource with attribute a', generating a contribution c and requiring time τ to complete the action. a', c and τ are all functions, which we can represent using the triplet $(a^M(t,a,d), c^M(t,a,d), \tau^M(t,a,d))$ (for notational compactness, we index these functions by time t instead of modeling the explicit dependence on K_t). We call $a^M(t,a,d)$ the *terminal attribute function*. Normally, we represent the costs and times using the vectors $c_{tad} = c^M(t,a,d)$ and $\tau_{tad} = \tau^M(t,a,d)$. We note as an aside that while we will usually model $(a^M(t,a,d), c^M(t,a,d), \tau^M(t,a,d))$ as $\mathcal{F}_t-$measurable, this is certainly not always the case. For example, section 4 describes an application in rail car distribution. In this application, empty freight cars are moved to customers to move loads of freight. The destination of a load is typically not known until the car is released loaded back to the railroad. The travel time of the movement is not known until the car actually reaches the destination.

The set $\mathcal{D}$ is the set of *types* of decisions we make. The decision vector itself is represented using:

$x_{tad} =$ The number of times that we act on a resource with attribute a using decision d at time t.
$x_t = (x_{tad})_{a \in \mathcal{A}, d \in \mathcal{D}}$.
$=$ The vector of decisions at time t.

Letting c_t similarly represent the vector of contributions at time t provides for a compact representation that matches standard modeling notation. Most transportation costs are

linear in the decision variables, and as a result, the total contribution at time t can be written as:

$$\begin{aligned} C_t(x_t) &= \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} c_{tad} x_{tad} \\ &= c_t x_t \end{aligned}$$

It is important to realize that our notation for stochastic problems is different in a subtle but important way than the notation conventionally used in deterministic transporation models. For example, it is normal to let x_{ijt} be the flow from location i to location j departing at time t. The index j effectively presumes a deterministic outcome of the decision (the notation $x_{ijt}(\omega)$ does not fix the problem; we would have to write $x_{i,j(\omega),t}$ which is quite ugly). We might not question the outcome of a decision to send a truck or plane from i to j (frequent fliers will remember at least one occasion when the plane did not arrive at the proper destination as a result of weather problems). But in more complex problems where we are capturing a larger vector of attributes, the terminal attribute function $a^M(t, a, d)$ cannot in general be assumed to be a deterministic function of (t, a, d). The representation of a decision using x_{tad} is important for stochastic problems since the variable is indexed only by information available when the decision is made.

For algebraic purposes, it is useful to define:

$$\begin{aligned} \delta_{t',a'}(t, a, d) &= \text{Change in the system at time } t' \text{ given a decision executed at time } t. \\ &= \begin{cases} 1 \text{ if } M_t(t, a, d) = (a', \cdot, t' - t) \\ 0 \text{ otherwise} \end{cases} \end{aligned}$$

We note that if d represents a decision to couple two resources, then a is the attributes of the resource, d contains the information about the resource being coupled with, and a' is the concatenation of two attribute vectors.

Using this notation, we can now write the dynamics of our resource variable (incorporating the time-lagging of information):

$$R_{t+1,a't'} = R_{t,a't'} + \hat{R}_{t+1,a't'}(\omega) + \sum_{d \in \mathcal{D}} \sum_{a \in \mathcal{A}} \delta_{t',a'}(t, a, d) x_{tad} \qquad a' \in \mathcal{A}, \quad t' > t \tag{2}$$

It is common in transportation problems to focus on decisions that move resources from one location to the next. While this is the most obvious dimension, it is important to capture other types of decisions.

Our notation for representing decisions offers considerable flexibility. It is a common misconception in the modeling of transportation systems that decisions always represent movements from one location to another. Examples of different classes of decisions other than spatial movements include: cleaning dirty vehicles, repairing or maintaining equipment, sending a driver off-duty, using outside contractors to perform a task, transferring rail cars from one shipper pool to another (this is a form of classification, and does not mean moving from one location to another), buying/selling/leasing equipment, and hiring/firing drivers.

In deterministic problems, decisions are made by solving a particular instance of an optimization problem. In stochastic problems, we have to capture the time staging of decisions and information. We represent the process of making decisions at time t using:

$I_t =$ The set of information available for making a decision.
$X_t^\pi(I_t) =$ The decision function of policy $\pi \in \Pi$ which returns a vector x_t given the information set I_t.

In section 3.6, we describe different classes of information, and the types of decision functions these produce.

For our problems, the decision function will be some sort of mathematical program, since the decisions typically are vectors, possibly of fairly high dimensionality. Later we provide specific examples of decision functions, but for now, we simply assume that they produce feasible solutions. The most important constraint that must be satisfied is flow conservation:

$$\sum_{d \in \mathcal{D}} x_{tad} = R_{ta} \quad \forall a \in \mathcal{A}$$

In addition, the flows must be nonnegative and, in many applications (virtually all involving operational problems in transportation) integer.

3.4 Modeling state variables

It is useful at this point to make a brief comment about "state variables," since these take on different meanings in different communities. In our modeling framework, the attribute vector a captures the "state" of a particular resource. $R_t = (R_{ta})_{a \in \mathcal{A}}$ is the "state" of the vector of resources. I_t (which we have not completely defined) is the "information state" of the system. In some subcommunities (notably, people who solve crew scheduling problems using column generation techniques), the management of multiple resources is decomposed into subproblems involving the optimization of a single resource. In this context, someone might talk about a large "state space" but refer to the attribute space of a single resource.

It is very common in the operations research literature (most commonly in the context of dynamic programming and Markov decision processes) to talk about the "state" of the system, where the state variable captures the amount of product being stored or the customer demands that have been backlogged. In this setting, the "state" of the system refers to the resource state variable, R_t. Even recently, discrete dynamic programming models have been proposed using R_t as the state variable. Not surprisingly, the number of possible realizations of R_t (assuming it is discrete) will be huge even for toy problems.

Of course, the real state variable must be what we know or, literally, the state of our knowledge, which we denote by K_t. Other authors refer to this as the *information state*. We let I_t be the information state, but claim that there are potentially four classes of information:

a) Knowledge - This is the data in the vector K_t, capturing the exogenous data that has been provided to the system.
b) Forecasts of exogenous processes - This is information from a forecasting model, representing projects of what might happen in the future. If we are making a decision at time t, this would be a projection of $(\hat{\xi}_{t+1}, \hat{\xi}_{t+2}, \ldots, \hat{\xi}_T)$. We may use a point forecast of future events, or forecast a set of future scenarios which would be represented using the set $\hat{\Omega}_t$ (the set of future events forecasted at time t). If $|\hat{\Omega}| = 1$, then we are using a traditional point forecast.
c) Forecasts of the impact of decisions now on the future. In this chapter, this dimension will be captured through the *recourse function* and hence we denote the set of possible recourse functions, estimated at time t (but capturing the impact on the future) by $\mathcal{Q}_t$.
d) Plans - These are projections of decisions to be made in the future, which can be expressed in a variety of ways (it is useful to think of these as *forecasts* of future decisions). A convenient way is to represent them as a vector of decisions $x^p_t = (x^p_{tt'})_{t' \geq t}$, where $x^p_{tt'}$ is the plan for time t' using the information available at time t. We note that plans are almost always expressed at some level of aggregation. Normally, we use plans as a guide

and penalize deviations from a plan.

The last three classes of information are all forms of forecasts. We assume that these are generated from data that is a function of K_t. However, while a forecast is generated from knowledge, they do not represent knowledge itself. All companies seek to improve decision-making by improving the knowledge base K_t, but they also consider the value of including forecasts (many transportation companies do not perform short term operational forecasts, and most research into problems such as dynamic vehicle routing does not use forecasts) or future plans. Companies make explicit decisions to add these classes of information to their decision making process (and adjust the process accordingly).

Using this definition of information, the information state can come in a variety of forms, such as $I_t = (K_t), I_t = (K_t, \hat{\Omega}_t), I_t = (K_t, x_t^p)$ and $I_t = (K_t, Q_t)$. Later we show that different classes of information give rise to the major classes of algorithms known in the operations research community. For the moment, it is necessary only to understand the different ways of representing the "state" of the system. Our notation contrasts with the standard notation S_t for a state variable. The problem is that S_t is not very explicit about what is comprising the state variable. We suggest using S_t when we want to refer to a generic "state," and use a, R_t, K_t or I_t when we want to express explicit dependence on, respectively, the attribute of a single resource, the resource state vector, the entire knowledge base, or a broader information set.

Using these notions of state variables, it is useful to revisit how we write our cost and decision functions. The representation of costs and decisions using the notation c_{tad} and x_{tad} suggests that both the costs and decisions are a function only of the attribute vector of the resource, although this does not have to be the case. We may write the decision function as $X^\pi(R_t)$ if all other types of information are static. The reader may write $X^\pi(K_t)$ to express the explicit dependence on the larger knowledge base, but this generality should be reserved for problems where there are parameters which are evolving over time, and whose values affect the forward evolution of the system.

3.5 The optimization problem

Our problem is to find a decision function X^π that solves the following expression:

$$F^* = sup_{\pi \in \Pi} E F^\pi \tag{3}$$

$$= sup_{\pi \in \Pi} E \left\{ \sum_{t \in \mathcal{T}} C_t(X_t^\pi(I_t)) \right\} \tag{4}$$

The system has to respect the following equations governing the physical and information dynamics:

Physical dynamics:

$$R_{t+1,a't'}(\omega) = R_{t,a't'}(\omega) + \hat{R}_{t+1,a't'}(\omega) + \sum_{d\in\mathcal{D}}\sum_{a\in\mathcal{A}} \delta_{t',a'}(t,a,d)x_{tad} \quad a' \in \mathcal{A}, \quad t' > t \tag{5}$$

Informational dynamics:

$$K_{t+1} = U^K(K_t, \xi_{t+1}) \tag{6}$$

The decision function X_t^π is assumed to produce a feasible decision. For this reason, flow conservation constraints and upper bounds are not included in this formulation.

The optimization problem is one of choosing a function. The structure of the decision function depends on the information available. Within an information class, a decision function is typically characterized by a family of parameters and we have to choose the best value for these parameters.

3.6 A brief taxonomy of problems

Using our modeling framework, we can provide a brief taxonomy of major problem classes that arise in transportation. We divide our taxonomy along the three major dimensions of resources, processes and controls.

Resources

By just using the attribute vector a notation, we can describe six major problem classes in terms of the resources being managed:

1) Basic inventory problems - $a = \{\}$ (no attributes). This is the classical single product inventory problem.
2) Multiproduct inventory problems - $a = \{k\}$ where $k \in \mathcal{K}$ is a product type.
3) Single commodity flow problems - $a = \{i\}$ where $i \in \mathcal{I}$ is a state variable (such as a city or geographical location).
4) Multicommodity flow problems - $a = \{i, k\}$ where $i \in \mathcal{I}$ is a state variable (such as a location) and $k \in \mathcal{K}$ is a commodity class.
5) Heterogeneous resource allocation problem - $a = \{a_1, a_2, \ldots, a_N\}$. In these more complex problems, it is possible to divide the attribute vector into static attributes, a^s, which do

not change over time, and dynamic attributes, a^d, which do change. Writing $a = \{a^s, a^d\}$, we can think of a^d as a resource state variable, and a^s as a resource type variable.

6) The multilayered resource allocation problem - $a = \{a^1|a^2|\cdots|a^L\}$ where a^c is the attributes of resource class c. Here, a is a concatenation of attribute vectors.

Although the sixth class opens the door to multilayered problems, it is useful to divide resource allocations between single layer problems, two layer problems (which most often involve an *active* resource layer representing people or equipment, and a *passive* layer representing customer requests), and multilayer problems.

We focus on single layer problems in this chapter, which include the first five types of attribute vectors. Of these, the first four are typically characterized by small attribute spaces, where it is possible to enumerate all the elements in $\mathcal{A}$, while heterogeneous resource allocation problems are typically characterized by an attribute space that is too large to enumerate. As we point out later, this creates special problems in the context of stochastic resource allocation problems.

System dynamics

Under the heading of system dynamics, we divide problems along three major dimensions:

1) The time staging of information - The two major problem classes are:
   a) Two-stage problems.
   b) Multistage problems.
2) Travel times (or more general, decision completion times). We define two major classes:
   a) Single-period times - $\tau_{tad} = 1$ for all $a \in \mathcal{A}, d \in \mathcal{D}$.
   b) Multiperiod times - $1 \leq \tau_{tad} \leq \tau^{max}$. We assume that $\tau_{tad} \geq 1$ but we can relax this requirement and model problems where $\tau_{tad} = 0$.
3) Measurability of the modify function. We again define two major classes:
   a) The function $M(t, a, d)$ is $\mathcal{F}_t-$measurable. This means that $(a^M(t, a, d), c^M(t, a, d), \tau^M(t, a, d))$ is deterministic given a, d and other parameters that are known at time period t.
   b) The function $M(t, a, d)$ is not $\mathcal{F}_t-$measurable. This is common, although we are not aware of any research addressing this issue.

Controls

We first divide problems into two broad classes based on control structure:

1) Single agent control structure - The entire company is modeled as being controlled by a single agent.
2) Multiagent control structure - We model the division of control between multiple agents.

Starting with the single agent control structure, we can organize problems based on the information available to make a decision. Earlier, we described four classes of information. We can now describe four classes of algorithms built around these information sets:

a) $I_t = (K_t)$ - This is our classic myopic algorithm, widely used in simulations. This is also the standard formulation used (both in practice and in the research community) for dynamic vehicle routing problems, and other on-line scheduling problems.
b) $I_t = (K_t, \hat{\Omega}_t)$ - If $|\hat{\Omega}_t| = 1$, this is our classical rolling horizon procedure using a point forecast of the future. This represents standard engineering practice for fleet management problems and other dynamic resource allocation problems. If $|\hat{\Omega}_t| > 1$, then we would obtain a scenario-based stochastic programming model. The use of these formulations for multistage problems in transportation and logistics is very limited.
c) $I_t = (K_t, x_t^p)$ - Here we are making decisions reflecting what we know now, but using plans to help guide decisions. This information set typically gives rise to proximal point algorithms, where the proximal point term penalizes deviations from plan.
d) $I_t = (K_t, Q_t)$ - This information set gives rise to dynamic programming formulations, Bender's decomposition and other methods for approximating the future. Typically, the recourse function Q_t is itself a function of a distributional forecast $\hat{\Omega}_t$, so it is appropriate to write $Q_t(\hat{\Omega}_t)$ to express this dependence.

This breakdown of different types of decision functions, each based on different types of information, nicely distinguishes engineering practice ($I_t = (K_t)$ or $I_t = (K_t, \hat{\Omega}_t)$ with $|\hat{\Omega}| = 1$) from the stochastic programming literature ($I_t = (K_t, \hat{\Omega}_t)$ with $|\hat{\Omega}| > 1$ or $I_t = (K_t, Q_t)$). The use of proximal point algorithms has been studied in the stochastic programming literature, but the use of plans (generated from prior data) to help guide future decisions is often overlooked in the modeling and algorithmic community. If stochastic programming is to gain a foothold in engineering practice (within the transportation and logistics community), it will be necessary to find the problem classes where the more advanced decision sets add value.

Complex problems in transportation, such as railroads, large trucking companies and the air traffic control system, are characterized by multiple decision-making agents. We would represent this structure by defining:

$\mathcal{D}_q =$ The subset of decisions over which agent q has control.
$I_{tq} =$ The information available to agent q at time t.

Then $X_{tq}^{\pi}(I_{tq})$ is the decision function for agent q given information I_{tq} at time t.

Multiagent systems capture the organization of information. By contrast, classical stochastic programming models focus on the flow of information. In transportation, modeling information is important, but we typically have to capture both the organization and flow.

We also find that in a multiagent system, we may have to forecast the behavior of another agent (who may work within the same company). This can be an important source of uncertainty in large operations.

4 A case study: freight car distribution

When moving freight by rail (for the purposes of this discussion, we exclude the movement of intermodal freight such as trailers and containers on flatcars), a shipper requests one or more cars, of a particular type, at his dock for a particular day. The request may be for one or two cars, or as many as 100 or more. The railroad identifies specific cars that can be assigned to the request, and issues a "car movement order" to get the car to the shipper. The car may be in a nearby yard, requiring only the movement of a "local" train to get the car to the shipper. Just as easily, the car may have to move from a much farther location through a sequence of several trains before arriving at the final destination.

Freight cars come in many types, often looking the same to the untrained eye but appearing very different to the shipper. For example, there are 30 types of open top gondola cars ("gons" in the industry). When a railroad cannot provide the exact type of car from the closest depot on the correct day, it may resort to three types of substitution:

1) Geographic substitution - The railroad may look at different sources of cars and choose a car that is farther away.
2) Temporal substitution - The railroad may provide a car that arrives on a different day.
3) Car type substitution - The railroad may try to satisfy the order using a slightly different car type.

Once the decision has been made to assign a car to a customer request, the railroad begins the process of moving a car to the destination. If the car is far away, this may require movements on several trains, passing through one or more intermediate *classification yards* which handle the sorting process. Travel times are long, and highly variable. It can take three weeks to move an empty car to a customer, wait for it to load, move it loaded, and then wait for it to unload (known as a car cycle). Travel times typically range between four to ten days or more. Travel times between a pair of locations that averages six days can see actual transit times between four and eight days.

From the perspective of car distribution, there are three important classes of dynamic information: the flow of customer requests for capacity, the process of cars becoming empty (either because a shipper has emptied and released the car or because another railroad has returned the car empty), and the travel times for cars moving from one location to another.

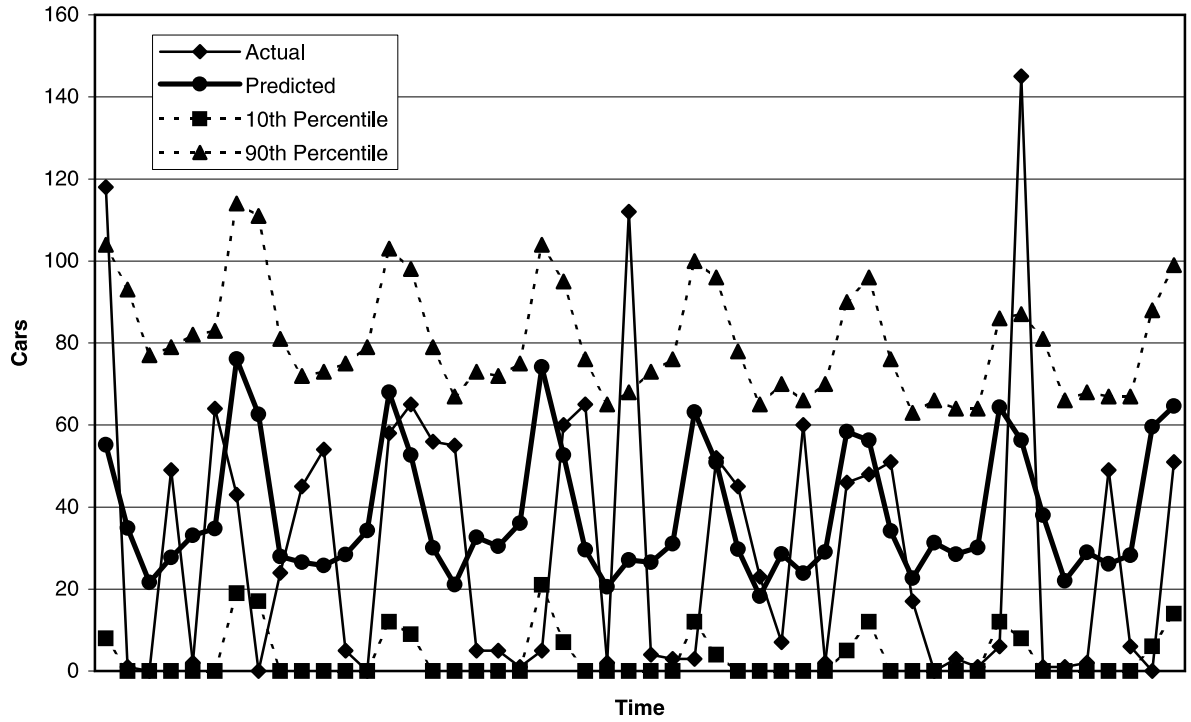

Fig. 1. Actual vs. predicted forecasts of future demands for empty cars, showing the 10^{th} and 90^{th} percentiles

Customer orders are typically made the week before the car is actually needed, but some orders are made more than a week in advance, and some orders are made at the last minute (especially from large, high priority customers). There is very little advance information about empty cars, and of course, transit times are only known after the movement is completed. Thus, we see information processes where the difference when a resource is knowable and actionable is large (customer orders), small (empty cars), and where the modify function is not $\mathcal{F}_t-$measurable.

It is useful to get a sense of the variability of the data. Figure 1 is an actual graph of the demand for cars at a regional level, showing actual, predicted, and both 10^{th} and 90^{th} percentiles. This graph ignores the presence of booked orders, and in practice, most orders are known a week into the future. For this reason, customer orders are not the largest source of uncertainty in an operational model. A much more significant source of error arises from the forecast of empty cars. Figure 2 shows a similar graph similar for a particular type of freight car at a specific location. We again see a large degree of variability. In this case, there is little advance information.

One of the most difficult sources of uncertainty arises in transit times. In railroads, it is not unusual to see transit times that range between five and ten days. This source of noise is

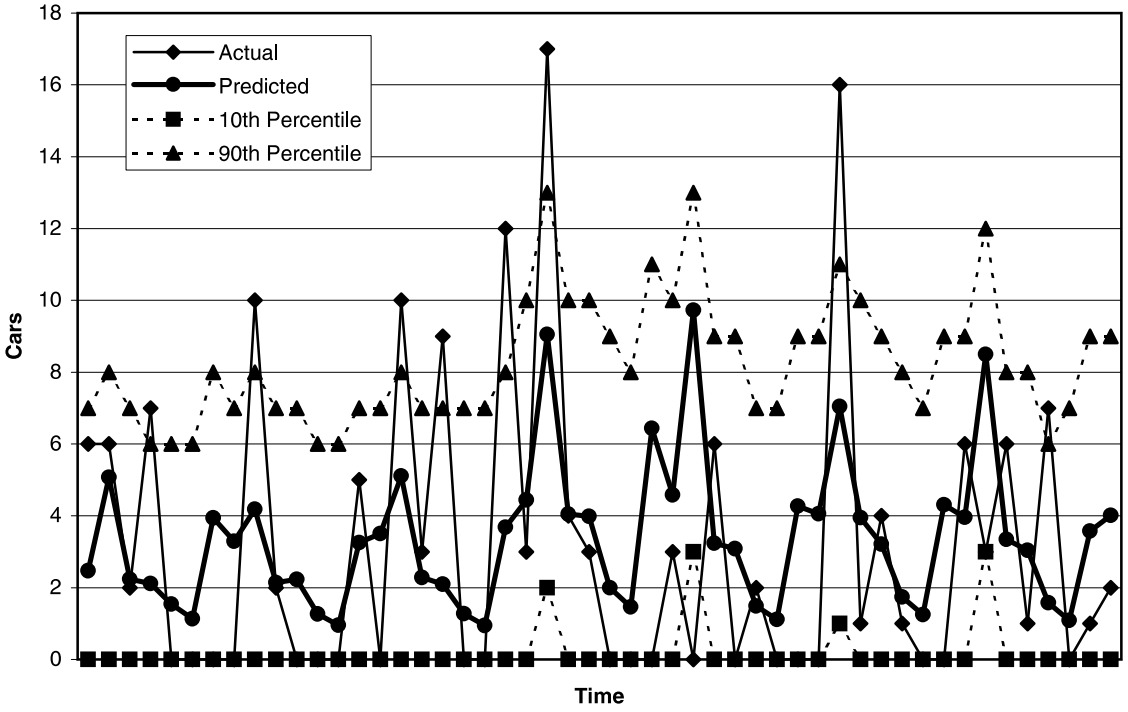

Fig. 2. Actual vs. predicted forecasts of supplies of empty cars, showing the 10^{th} and 90^{th} percentiles

particularly problematic. It means that if we ship 10 cars from i to meet a demand at j, we are not sure when they will arrive. It has been suggested that we can improve our forecast of empty cars becoming available by using what we know about cars that are currently moving loaded (we know where they are going, so if we could estimate the transit time, we could estimate when they are becoming available). The uncertainty of transit times complicates this analysis.

We are now ready to consider more carefully the decision classes that govern the problem. As a result of the long travel times and high degree of uncertainty, it is not possible to simply wait until orders become known before a car is assigned to satisfy the order. The situation is further complicated by the fact that they cannot always let a car sit until there is an order to assign it to. A car may become available at a location that does not have the capacity to store the car. As a result, the railroad faces four possible classes of decisions when a car becomes empty:

1) Send it directly to a customer who has booked an order. Normally, we assume that this decision is to assign a car to a specific order, but it could be modified to send the car to a customer (where it would be assigned to a specific order after it arrives).
2) Send it to a *regional depot* which only serves customers in the region.
3) Send it to a *classification yard* where cars can be sorted and moved out on different

trains. A classification yard at a railroad is a major facility and represents a point where it is easiest to make a decision about a car. From a classification yard, a car may be sent to another classification yard, a regional depot or directly to a customer.

4) Do nothing. This means storing the car at its current location. This is generally not possible if it just became available at a customer, but is possible if it is at a storage depot.

Not every car can be immediately assigned to an order, partly because some orders simply have not been booked yet, and possibly because there are times of the year when there are simply more cars than we need. At the same time, one would expect that we do not always assign a car to a particular order, because not all the available cars are known right now. However, there is a strong bias to find an available car that we know about right now (even if it is a longer distance from the order) than to use a car that might become available later.

5 The two-stage resource allocation problem

We start with the two-stage problem because it is fundamental to multistage problems, and because some important algorithmic issues can be illustrated with minimum complexity. It should not be surprising that we are going to solve multistage problems basically by applying our two-stage logic over and over again. For this reason, it is particularly important that we be able to understand the two-stage problem very well.

We begin our presentation in section 5.1 with a brief discussion of our notational style. Two-stage problems are relatively simple, and it is common to use notational shortcuts to take advantage of this simplicity. The result, however, is a formulation that is difficult to generalize to harder problems. 5.2 summarizes some of the basic notation used specifically for the car distribution problem. We introduce our first model in section 5.3 which presents models that are in practice today. We then provide three levels of generalization on this basic model. The first (section 5.4) introduces uncertainty without any form of substitution, producing the classical "stochastic programming with simple recourse" formulation. The second models the effect of regional depots (section 5.5), which produces a separable two-stage problem which can be solved using specialized techniques. The last model considers classification yards which requires modeling general substitution (section 5.6), and brings into play general two-stage stochastic programming, although we take special advantage of the underlying network structure. Finally, section 5.7 discusses some of the issues that arise for problems with large attribute spaces.

5.1 Notational style

One of the more subtle modeling challenges is the indexing of time. In a two stage problem, this is quite simple. Often, we will let x denote an initial decision, followed by new information (say, ξ), after which there is a second decision (perhaps denoted by y) that is allowed to use the information in the random variable ξ.

This is very simple notation, but does not generalize to multistage problems. Unfortunately, there is not a completely standard notation for indexing activities over time. The problem arises because there are two processes: the information process, and the physical process. Within the information process, there is exogenous information, and the process of making decisions (which can be viewed as endogenously controllable information). In many problems, and especially true of transportation, there is often a lag between the information process (when we know about an activity) and the physical process (when it happens). (We ignore a third process, which is the flow of financial rewards, such as billing a customer for an activity at the end of a month.)

In the operations research literature, it is common to use notation such as x_t to represent the vector of flows occurring (or initiating) in time t. This is virtually always the case in a deterministic model (which ignores completely the time staging of information). In stochastic models, it is more common (although not entirely consistent) to index a variable based on the information content. In our presentation, we uniformly adopt the notation that any variable indexed by time t is able to use the exogenous information up through and including time t (that is, $\xi_0, \xi_1, \ldots, \xi_t$). If x_t is a decision made in time t, then it is also allowed to see the information up through time t. It is often useful to think of ξ_t as information arriving "during time period t" whereas the decision x_t is a function determined at the end of time period t.

We treat $t = 0$ as the starting point in time. The discrete time $t = 1$ refers to the time interval between 0 and 1. As a result, the first set of new information would be ξ_1. If we let S_0 be our initial state variable, we can make an initial decision using only this information, which would be designated x_0. A decision made using ξ_1 would be designated x_1.

There may be a lag between when the information arrives about an activity and when the activity happens. It is tempting, for example, to let D_t be the demands that arrive in period t, but we would let D_t be the orders that become known in time period t. If a customer calls in an order during time interval t which has to be served during time interval t', then we would denote this variable by $D_{tt'}$. Similarly, we might make a decision in time period t to serve an order in time period t'; such an activity would be indexed by $x_{tt'}$.

A more subtle notational issue arises in the representation of state variables. Here we depart from standard notation in stochastic programming which typically avoids an explicit definition of a state variable (the "state" of the system going into time t is the vector of decisions made in the previous period x_{t-1}). In resource allocation problems, vectors such as x_t can have a very large number of dimensions. These decisions produce future inventories of resources which can be represented using much lower dimensional state variables. In practice, these are much easier to work with.

It is common in multistage problems to let S_t be the state of the system at the beginning of time period t, after which a decision is made, followed by new information. Following our convention, S_t would represent the state after the new information becomes known in period t, but it is ambiguous whether this represents the state of the system before or after a decision has been made. It is most common in the writing of optimality equations to define the state of the system to be all the information needed to make the decision x_t. However, for computational reasons, it is often useful to work in terms of the state of the system immediately after a decision has been made. If we let S_t^+ be the *complete* state variable, giving all the information needed to make a decision, and let S_t be the state of the system immediately after a decision is made, the history of states, information and decisions up through time t would be written:

$$h_t = \{S_0^+, x_0, S_0, \xi_1, S_1^+, x_1, S_1, \xi_2, S_2^+, x_2, S_2, \ldots, \xi_t, S_t^+, x_t, S_t, \ldots\} \tag{7}$$

We sometimes refer to S_t as the *incomplete* state variable, because it does not include the information ξ_{t+1} needed to determine the decision x_{t+1}. For reasons that are made clear later (see section 6.2), we find it more useful to work in terms of the incomplete state variable S_t (and hence use the more cumbersome notation S_t^+ for the complete state variable).

In this section, we are going to focus on two-stage problems, which consist of two sets of decision vectors (the initial decision, and the one after new information becomes known). We do not want to use two different variables (say, x and y) since this does not generalize to multistage problems. It is tempting to want to use x_1 and x_2 for the first and second stage, but we find that the sequencing in equation (7) better communicates the flow of decisions and information. As a result, x_0 is our "first" stage decision while x_1 is our second stage decision.

Given the complexity of the problem, the simplicity of the models in engineering practice is amazing. As of this writing, we are aware of two basic classes of models in use in North America: myopic models, which match available cars to orders that have already been booked into the system, and models with deterministic forecasts, which add to the set of known orders additional orders that have been forecasted. We note that the railroad that uses a purely myopic model is also characterized by long distances, and probably has customers which, in response to the long travel times, book farther in advance (by contrast, there is no evidence that even a railroad with long transit times has any more advance information on the availability of empty cars). These models, then, are basically transportation problems, with available cars on the left side of the network and known (and possibly forecasted) orders on the right side.

The freight division of the Swedish National Railroad uses a deterministic time-space network to model the flows of loaded and empty cars and explicitly models the capacities of trains. However, it appears that the train capacity constraints are not very tight, simplifying the problem of forecasting the flows of loaded movements. Also, since the model is a standard, deterministic optimization formulation, a careful model of the dynamics of information has not been presented, nor has this data been analyzed.

The car distribution problem involves moving cars between the locations that handle cars, store cars and serve customers. We represent these using:

$\mathcal{I}^c$ = Set of locations representing customers.
$\mathcal{I}^{rd}$ = Set of locations representing regional depots.
$\mathcal{I}^{cl}$ = Set of locations representing classification yards.

It is common to represent the "state" of a car by its location, but we use our more general attribute vector notation since it allows us to handle issues that arise in practice (and which create special algorithmic challenges for the stochastic programming community):

$\mathcal{A}^c$ = The set of attributes of the cars.
$\mathcal{A}^o$ = The set of attributes of an order, including the number of days into the future on which the order should be served (in our vocabulary, its actionable time).
$R^c_{t,at'}$ = The number of cars with attribute a that we know about at time t that will be available at time t'. The attribute vector includes the location of the car (at time t') as well as its characteristics.
$R^o_{t,at'}$ = The vector of car orders with attribute $a \in \mathcal{A}^o$ that we know about at time t which are needed at time t'.

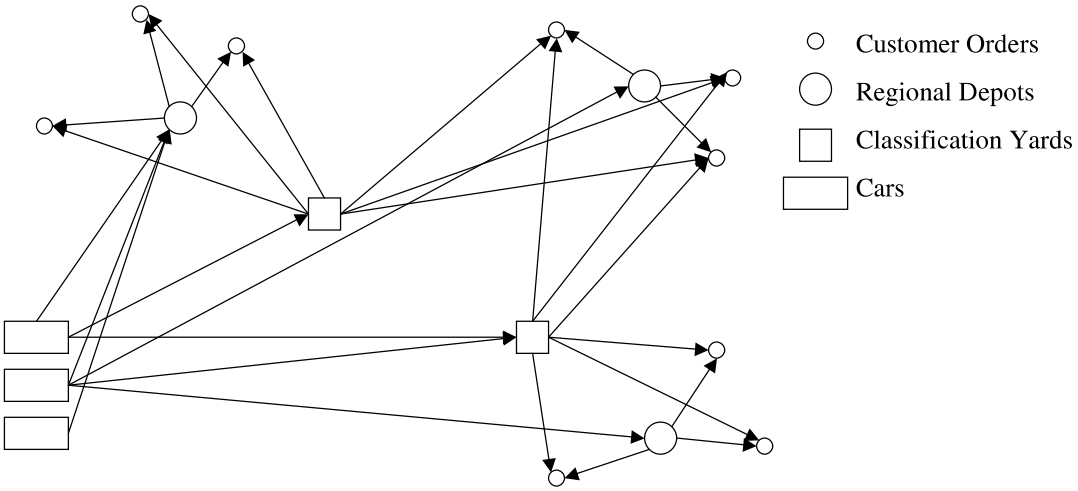

Fig. 3. Car distribution through classification yards

Following the notational convention in equation (7), We let $R_0^{+,c}$ and $R_0^{+,o}$ be the initial vectors of cars and orders at time 0 before any decisions have been made, whereas R_0^c and R_0^o are the resource vectors after the initial decision x_0 has been implemented.

It is common to index variables by the location. We use a more general attribute vector a, where one of the elements of an attribute vector a would be the location of a car or order. Rather than indexing the location explicitly, we simply make it one of the attributes.

The decision classes are given by:

$\mathcal{D}^c =$ The decision class to send cars to specific customers, where $\mathcal{D}^c$ consists of the set of customers (each element of $\mathcal{D}^c$ corresponds to a location in $\mathcal{I}^c$).

$\mathcal{D}^o =$ The decision to assign a car to a type of order. Each element of $\mathcal{D}^0$ corresponds to an element of $\mathcal{A}^o$. If $d \in \mathcal{D}^o$ is the decision to assign a car type, we let $a_d \in \mathcal{A}^o$ be the attributes of the car type associated with decision d.

$\mathcal{D}^{rd} =$ The decision to send a car to a regional depot (the set $\mathcal{D}^{rd}$ is the set of regional depots - we think of an element of $\mathcal{I}^{rd}$ as a regional depot, while an element of $\mathcal{D}^{rd}$ as a decision to go to a regional depot).

$\mathcal{D}^{cl} =$ The decision to send a car to a classification yard (each element of $\mathcal{D}^{cl}$ is a classification yard).

$d^\phi =$ The decision to hold the car ("do nothing").

The different decision classes are illustrated in figure 3, where a car can be shipped directly to a customer, a regional depot, or a classification yard.

Our complete set of decisions, then, is $\mathcal{D} = \mathcal{D}^c \cup \mathcal{D}^o \cup \mathcal{D}^{rd} \cup \mathcal{D}^{cl} \cup d^\phi$. We assume that

we only act on cars (cars are the only *active* resource class, whereas orders are referred to as a *passive* resource class). We could turn orders into an active resource class if we allowed them to move without a car (this would arise in practice through outsourcing of transportation). Of these, decisions in $\mathcal{D}^o$ are constrained by the number of orders that are actually available. As before, we let x_{tad} be the number of times that we apply decision d to a car with attribute a given what we know at time t.

The contribution function is:

c_{tad} = The contribution from assigning a car with attribute a to an order for cars of type $d \in \mathcal{D}^o$, given what we know at time t. If $d \in \mathcal{D}^o$, then we assume that the contribution is a "reward" for satisfying a customer order, minus the costs of getting the car to the order. For all other decision classes, the contributions are the negative costs from carrying out the decision.

Since all orders have to be satisfied, it is customary to formulate these models in terms of minimizing costs: the cost of moving a car from its current location to the customer, and the "cost" of assigning a particular type of car to satisfy the order. Since rail costs are extremely complex (what is the marginal cost of moving an additional empty car on a train?), all costs are basically surrogates. The transportation cost could be a time or distance measurement. If we satisfy the customer order with the correct car type, then the car type cost might be zero, with higher costs (basically, penalties) for substituting different car types to satisfy an order. Just the same, we retain our maximization framework because this is more natural as we progress to more general models (where we maximize "profits" rather than minimize costs).

5.3 Engineering practice - Myopic and deterministic models

The most basic model used in engineering practice is a myopic model, which means that we only act on the vectors $R^c_{0t'}$ and $R^o_{0t'}$ (we believe that in practice, it is likely that companies even restrict the vector of cars to those that are actionable now, which means R^c_{00}). We only consider decisions based on what we know now (x_{0ad}), and costs that can be computed based on what we know now (c_{0ad}). This produces the following optimization problem:

$$\min_x \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} c_{0ad} x_{0ad} \tag{8}$$

subject to:

$$\sum_{d \in \mathcal{D}} x_{0ad} = R^c_{0a} \quad a \in \mathcal{A} \tag{9}$$

$$\sum_{a \in \mathcal{A}} x_{0ad} \leq R^o_{0a_d} \quad d \in \mathcal{D}^o \tag{10}$$

$$x_{0ad} \in Z_+ \tag{11}$$

Equation (10) restricts the total assignment of all car types to a demand type $a_d, d \in \mathcal{D}^o$, by the total known demand for that car type across all actionable times. The model allows a car to be assigned to a demand, even though the car may arrive after the time that the order should have been served. Penalties for late service are assumed to be captured in c_{0ad}.

It is easy to pick this model apart. First, the model will never send a car to a regional depot or classification yard (unless there happens to be a customer order at precisely that location). Second, the model will only send a car to an order that is known. Thus, we would not take a car that otherwise has nothing to do and begin moving to a location which is going to need the car with a high probability. Even worse, the model may move a car to an order which has been booked, when it could have been moved to a much closer location where there probably will be an order (but one has not been booked as yet). If there are more cars than orders, then the model provides almost no guidance as to where cars should be moved in anticipation of future orders,

Amidst these weaknesses are some notable strengths. First, the model is simple to formulate and solve using commercial solvers. Second, the model handles all three types of substitution extremely easily (especially important is substitution across time, something that models often struggle with). But, perhaps the most important feature is that the solution is easy to understand. The most overlooked limitation of more sophisticated models is that their solutions are hard to understand. If the data were perfect, then we would argue that the user should simply trust the model, but the limitations of the data preclude such a casual response.

The first generalization used in practice is to include forecasts of future orders, which we would represent using the vector $R^o_{tt'}$ for $t \in \mathcal{T}^{ph}$, where $\mathcal{T}^{ph}$ is the set of time periods in our planning horizon. The details of the process of forecasting future orders are, of course, not documented. The process of forecasting would generally have to be made at some level of aggregation (daily/weekly, customer level or regionl, and the car class). Particularly tricky is handling the time staging of orders. If a forecast is generated for a particular time t' in the future (using, for example, standard time series forecasting techniques applied to a historical dataset showing customer orders by time period), then we would be forecasting the total orders for time t', and then adding in the orders to be satisfied at time t' that are known now. We assume that we have a forecast $R^o_{tt'}$ representing the orders that would be placed at time t to be satisfied at time t'.

We let $R^o_{tt'}$ be a point forecast of future demands for $t \geq 1, t' \geq t$, with $R^o_{0t'}$, as before, the orders we know about now. We could also make a forecast of cars that will become available in the future, but this is still not normally done. As a result, our model using a deterministic forecast is given by:

$$\min_x \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} c_{0ad} x_{0ad} \tag{12}$$

subject to:

$$\sum_{d \in \mathcal{D}} x_{0ad} = R^c_{0a} \quad a \in \mathcal{A} \tag{13}$$

$$\sum_{a \in \mathcal{A}} x_{0ad} \leq \sum_{t \in \mathcal{T}^{ph}} R^o_{ta_d} \quad d \in \mathcal{D}^o \tag{14}$$

$$x_{0ad} \in Z_+ \tag{15}$$

Equation (14) includes demands that are known now (R^o_{0a}) and all orders that are forecasted to become known within the planning horizon. Note that we are using forecasted orders, but not forecasted cars. One model in industrial practice separately forecasts future cars becoming available, but these forecasts are independent of decisions being made now. To model this process, we would replace equation (13) with:

$$\sum_{d \in \mathcal{D}} x_{0ad} = \sum_{t \in \mathcal{T}^{ph}} R^c_{ta} \quad a \in \mathcal{A}$$

This approach assumes we are using an exogenous forecast of cars in the future, which ignores the impact of current decisions on future car supplies.

It would be possible to use a deterministic, time staged model over a planning horizon, but this would actually be fairly hard to solve, since it would be a moderately large integer multicommodity flow problem with time windows on the loads (it is the time windows that really complicates the formulation).

Models that incorporate forecasted demands have the immediate advantage of providing recommendations for cars which would otherwise not be assigned in a myopic model. The model will send cars to locations which normally book new orders, allowing the railroad to start the process of moving the car, rather than waiting until the last minute. Since we are only using a point forecast, the model will not be able to send cars to a location where they *might* be needed. This can be a problem when we are in a period where there is excess supply. This model can recommend letting cars sit at a location where there is

absolutely no chance of them being used, rather than moving them to a location where they might be used.

Our model does not include forecasts of empty cars. The common rationale for leaving these forecasts out is that they are so uncertain (it is not unusual for practitioners to ignore information which cannot be reasonably approximated by a point forecast). It also ignores many other operational issues such as train capacities (which we have already identified to be highly uncertain), yard capacities (which determines how many cars can be stored at a location) or the value of cars at the end of the horizon (which we could overcome with a multistage model).

There are a number of limitations of these simple models, but we would argue that a serious practical limitation is that the model will never recommend sending a car to a regional depot or classification yard. In one application with which we are familiar, the model will recommend sending a car to a particular customer (perhaps to serve a forecasted order). In the process of routing the car to the customer, the car will have to go through a regional depot. The railroad will then route the car to the depot, reoptimizing the assignment of the car to new orders as they become available. This is a highly heuristic way of accounting for uncertainty.

5.4 No substitution - a simple recourse model

Our first effort o incorporate uncertainty is a simple recourse model where we replace the decision class to assign cars to a specific *order* and instead allow us to send cars to a particular *customer* (or equivalently, to a customer location). The difference is that if a customer only places one order, then in the first model we can only send him one car. In our simple recourse model, we may send more cars to the customer location at time t than has been ordered at time t' in the hopes that new orders will come in later. For this case, we define:

$R^o_{t,ct'}$ = The number of orders for customer c that we first learn about at time t that are actionable (must be served) at time t'.
$R^o_{t,c}$ = All the orders for customer c known at time t.
$= (R^o_{t,ct'})_{t' \geq t}$.

Of course, $\mathcal{R}^o_0$ are the orders we know about now, while $(\mathcal{R}^o_t)_{t>0}$ are the forecasted orders for the future. Unlike our first models, we are now going to explicitly model the uncertainty in the forecast of future orders. Looking at equation (14), we see that we only need the total forecast of future demands. For this reason, it is simpler to define:

$$\bar{R}^o_{1,c} = \sum_{t \in \mathcal{T}^{ph} \setminus \{0\}} \sum_{t' \in \mathcal{T}^{ph} \setminus 0} R^o_{t,ct'}$$

$\bar{R}^o_{1c}$ is a random variable representing all "future" demands, which would be derived from a forecasting model. Note that we have aggregated not only on all orders that would become known in the future (t), but we have also aggregated across the dates when the orders would need to be satisfied (t'). Let:

R^c_{0,i_d} = The number of cars (indicated by the superscript c) sent to customer i_d, $d \in \mathcal{D}^c$, where the decision is made at time 0 but the cars can be used at time 1 (the second stage).
= $\sum_{a \in \mathcal{A}} x_{0ad}$

The decisions x_{0ad} must be made before the orders $(R^o_{ti})_{t>0}$ become known. In our simple recourse model, we assume that a car sent to customer c cannot then, at a later time, be sent to another customer. It is either used to satisfy an order (within our planning horizon) or it sits idle. Let:

c^o_i = The (positive) contribution from satisfying an order for customer $i \in \mathcal{I}^c$.
c^h_i = The contribution (typically negative) from sending a car to customer i and then having it sit.

Now let:

x^o_{1i} = The number of cars assigned to serve an order (after they arrive at customer i).
x^h_{1i} = The number of cars that are held at customer i.

x^o_1 and x^h_1 are random variables defined by:

$$x^o_{1c}(R^c_{0,c}, \bar{R}^o_{1c}(\omega)) = \min\{R^c_{0,c}, \bar{R}^o_{1c}(\omega)\}$$
$$x^h_{1c}(R^c_{0,c}, \bar{R}^o_{1c}(\omega)) = \max\{0, R^c_{0,c} - \bar{R}^o_{1c}(\omega)\}$$

We should note that our choices for $x^o_{1c}(R^c_{0,c}, \bar{R}^o_{1c}(\omega))$ and $x^h_{1c}(R^c_{0,c}, \bar{R}^o_{1c}(\omega))$ seem a bit obvious, but they are in fact the result of a trivial optimization problem.

$R^c_{0,c1}$ is a function of the first stage decisions x_0. Given x_0, the expected second stage reward is given by:

$$
\begin{aligned}
\bar{C}_{0,1}(R_0^c(x_0)) &= \text{Expected costs using the information available in time period 0} \\
&\quad \text{that would be incurred in time period 1.} \\
&= E\left\{\sum_{c\in\mathcal{I}^c}\left(c_c^o x_{1c}^o(R_{0c}^c(x_0),\bar{R}_{1c}^o)+c_c^h x_{1c}^h(R_{0c}^c,\bar{R}_{1c}^o)\right)\right\} \\
&= \sum_{c\in\mathcal{I}^c}\left(\bar{C}_{0,c1}(R_{0c}^c(x_0))\right)
\end{aligned}
$$

The functions $\bar{C}_{0,c1}(R_{0c}^c)$ are concave. If the random demands are discrete, then it is also possible to show that $\bar{C}_{0,c1}(R_{0c}^c)$ is piecewise linear, concave, with the breakpoints at integer values of R_{0c}^c. Since these functions are computed as expectations of scalar random variables, computing them is quite easy once the distribution of demands is known. Of course, forecasting future demands is in practice fairly tricky, primarily because of the process of customers booking orders in advance.

We can now formulate our problem as follows:

$$
\min_{x_0}\left\{c_0x_0+\bar{C}_{0,1}(R_0^c(x_0))\right\} \tag{16}
$$

subject to:

$$
\sum_{d\in\mathcal{D}} x_{0ad} = R_{0a}^{+,c} \quad a\in\mathcal{A} \tag{17}
$$

$$
\sum_{a\in\mathcal{A}} x_{0ad} \le R_{0a_d}^{+,o} \quad d\in\mathcal{D}^o \tag{18}
$$

$$
x_{0ad}\in Z_+ \tag{19}
$$

This is a convex nonlinear programming problem with network constraints. If the demands are discrete, producing piecewise-linear concave reward functions for each shipper, then we can use a standard trick for converting these problems into pure networks, as shown in figure 4.

5.5 Shipping to regional depots - a separable recourse model

The major weakness of the simple recourse model is that it does not capture the ability of the railroad to send cars to a regional depot, and then wait until the last minute to send cars from the depot to the customer. In fact, it is generally not possible to send a car to a customer unless the customer has specifically asked for the car. A more realistic model is to assume that the car has been sent to a local yard (which we refer to as a regional depot) where it is stored waiting for customers.

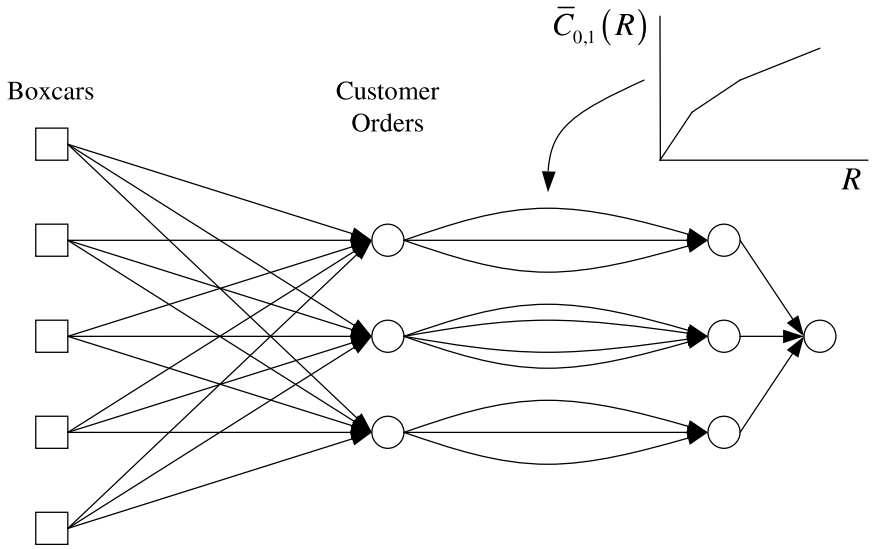

Fig. 4. The simple recourse problem as a pure network

In this section we present a more general model which captures the ability of a railroad to send cars to a regional depot, after which it can be distributed to customers. We must, however, introduce one key simplification (which we relax later), which is that while we can model general substitution rules between car types and order types in the first stage, we are not going to allow any substitution between car types in the second stage. One way to mitigate this approximation is to aggregate car types into more aggregate categories, and then assume that there is no substitution between major car categories.

We show in this section how we can solve this more general model, producing a solution that requires solving a network with the same structure as that produced for the case of simple recourse (figure 4). We begin by assuming that the demands from different shippers are statistically independent, and then present a more general result which uses a technique that we will use for harder problems.

5.5.1 The case of independent demands - an exact result

We begin by setting up some notation that we need for both models. For this work, it is easier to index decisions and contributions by the spatial location. This notation is clearer, although not as general.

For each regional depot there is a set of customers in this region. We represent this using:

$\mathcal{I}_r^c =$ The subset of customers in region $r \in \mathcal{I}^{rd}$. We further assume that customers in $\mathcal{I}_r^c$

can only be served using box cars at depot r.

x_{1ri} = The number of cars sent from $r \in \mathcal{I}^{rd}$ to $i \in \mathcal{I}^c_r$ to satisfy customer orders that become known in the second stage (here the destination i plays the role of the decision d in our earlier notation).

c_{1ri} = The contribution from sending cars from $r \in \mathcal{I}^{rd}$ to $i \in \mathcal{I}^c_r$ to satisfy customer orders that become known in the second stage.

R^o_{1c} = Random variable giving the number of orders for customer c in the second stage.

$R^c_{0r}(x_0)$ = Total number of cars sent to region r as a result of decisions made in the first period.

Both x_1 and R^o_1 are random variables. For a given realization of the second stage orders, we would find ourselves solving:

$$Q(R^c_{0r}, R^o_{1r}(\omega)) = \max_{x_1} \sum_{r \in \mathcal{I}^{rd}} \sum_{i \in \mathcal{I}^c_r} c_{1ri} x_{1ri}(\omega) \tag{20}$$

subject to:

$$\sum_{i \in \mathcal{I}^c_r} x_{1ri}(\omega) + x_{1rd^\phi}(\omega) = R^c_{0r} \quad \forall r \in \mathcal{I}^{rd} \tag{21}$$

$$x_{1ri}(\omega) \leq R^o_{1i}(\omega) \quad \forall i \in \mathcal{I}^c_r, r \in \mathcal{I}^{rd} \tag{22}$$

$$x_{1ri}(\omega) \geq 0 \quad \forall i \in \mathcal{I}^c_r, r \in \mathcal{I}^{rd} \tag{23}$$

where, as a reminder, d^ϕ is the "do nothing" decision. Problem (20)-(23) decomposes by regional depot, where the problem for each regional depot is easily solved as a sort. For a given region $r \in \mathcal{I}^c_r$, assume that

$$c^o_1 \geq c^o_2 \geq \ldots \geq c^o_{|\mathcal{I}^c_r|} \geq c^h_r$$

where $|\mathcal{I}^c_r|$ is the number of customers in region r. We have ordered the customers so that customer 1 is the most attractive, 2 is the second most attractive, and we have assumed that satisfying any customer is better than doing nothing (this assumption is easy to relax). Clearly, we would like to assign as much capacity as possible to the most valuable customers. We want to find the expectation of $E[Q_r(R^c_{0r}, R^o_{1r})] = Q_r(R^c_{0r})$. We are in particular interested in the slopes $Q_R(R^c_{0r}+1) - Q_r(R^c_{0r})$, since these form the coefficients on the arcs which give the marginal value of each additional unit of flow. We solve this using the following simple observation. Let $s = R^c_{0r}$, and let $\mathcal{E}(s,i)$ be the event that results in the s^{th} unit of flow being assigned to the $i^{th}-$most valuable customer. Define:

$\bar{R}^o_1(J)$ = $\sum_{j=1}^J R^o_{1,j}$

= Cumulative number of orders made by the top J customers.

The probability of the event $\mathcal{E}(s, J)$, then, is given by:

$$\begin{aligned} Prob[\mathcal{E}(s,J)] &= Prob[(\bar{R}_1^o(J-1) < s) \cap (\bar{R}_1^o(J) \geq s)] \\ &= Prob[\bar{R}_1^o(J-1) < s] + Prob[\bar{R}_1^o(J) \geq s] \\ &\quad - Prob[(\bar{R}_1^o(J-1) < s) \cup (\bar{R}_1^o(J) \geq s)] \end{aligned} \tag{24}$$

The events $(\bar{R}_1^o(J-1) < s)$ and $(\bar{R}_1^o(J) \geq s)$ are collectively exhaustive, so the last probability in equation (24) is equal to one. This allows us to reduce (24) to:

$$\begin{aligned} Prob[\mathcal{E}(s,J)] &= Prob[\bar{R}_1^o(J-1) < s] - (1 - Prob[\bar{R}_1^o(J) \geq s]) \\ &= Prob[\bar{R}_1^o(J-1) < s] - Prob[\bar{R}_1^o(J) < s] \end{aligned}$$

Thus, the probability that the s^{th} unit of flow is assigned to the J^{th} option is simply the difference between two cumulative distributions. These are easy to compute if the demands across customers are independent. Now let $v_r(s)$ be the expected value of the s^{th} unit of flow in depot r, given by:

$$v_r(s) = \sum_{i \in \mathcal{I}_r} c_i^o Prob[\mathcal{E}(s,i)] + c_r^h \left(1 - \sum_{i \in \mathcal{I}_r} Prob[\mathcal{E}(s,i)]\right)$$

The values $v_r(s)$ give the expected marginal value of each additional unit of flow sent into a regional depot.

Using the marginal values $v_r(s)$, our first stage problem is again a pure network very similar to the one used for simple recourse, but now with the property that the decision to send flow to a regional depot is considered explicitly. Our model will now send cars either directly to customers (to serve orders that have already been booked) or to regional depots for later assignment to orders that become known in the future.

Earlier, we considered the problem where we would send cars directly to the customer before knowing the customer demand. We would then incur an overage or underage penalty after learning the outcome. This strategy is referred to as simple recourse. In this section, we send a car to a regional depot; then, after we learn the demand, we decide which customers to allocate cars to. Since we are assigning cars from a single node over several links, this strategy has been referred to as *nodal recourse*.

Our analysis has been simplified in part by the assumption that the demands are independent (making it possible to find the partial cumulative distributions) and to an even greater degree by the assumption that each customer can be served by a single regional

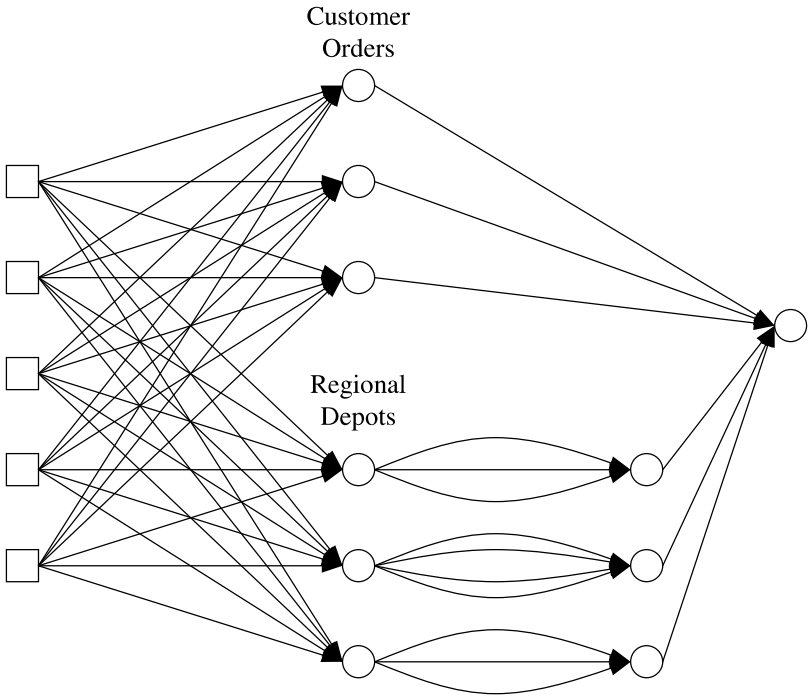

Fig. 5. The separable recourse problem as a pure network

depot. We first generalize our analysis to relax the assumption of independent demands, where we use a technique that will also allow us to relax the assumption that each customer is served by a single regional depot.

5.5.2 The general case - Monte Carlo methods

We have seen that in both the simple recourse case and the regional depot (nodal recourse) case, the problem reduces to finding piecewise linear, concave functions characterizing the value of cars at a location. Now we are going to introduce another technique for estimating these concave functions based on Monte Carlo sampling, which does not require making any independence assumptions between the demands of different customers.

Our second stage problem consists of finding:

$$Q(R_0^c) = EQ(R_0^c, R_1^o) \tag{25}$$

Our strategy is to solve this iteratively. At each iteration, we would choose an outcome ω. For this outcome, the conditional second stage function is given by:

$$Q(R_0^c, R_1^o(\omega)) = \max_{x_1(\omega)} \sum_{r \in \mathcal{I}^{rd}} \sum_{i \in \mathcal{I}_r^c} c_{1ri} x_{1ri}(\omega) \tag{26}$$

subject to:

$$\sum_{i \in \mathcal{I}_r^c} x_{1ri}(\omega) + x_{1rd^\phi}(\omega) = R_{0r}^c \quad \forall r \in \mathcal{I}^{rd} \tag{27}$$

$$x_{1ri}(\omega) \leq R_{1i}^o(\omega) \quad \forall i \in \mathcal{I}_r^c,\ r \in \mathcal{I}^{rd} \tag{28}$$

$$x_{1ri}(\omega) \geq 0 \quad \forall r \in \mathcal{I}^{rd}, i \in \mathcal{I}_r^c \tag{29}$$

Equations (26)-(29) are pretty easy to solve for a sample realization. Let $\hat{q}_{1r}(\omega)$ be the dual variable for constraint (27), reflecting the marginal value of another car. We would like to use this sample gradient information to build an approximation of $Q(R_0^c)$. The simplest strategy, of course, is to build a linear approximation of the form:

$$\hat{Q}(R_0^c) = \hat{q} \cdot R_0^c \tag{30}$$

but these are notoriously unstable. Although techniques are available to help these techniques (proximal point strategies, auxiliary functions), we are going to try to build a nonlinear function similar to the exact functions that we have seen so far. The simplest that we have seen starts with a piecewise linear function and then "tilts" it using stochastic subgradients. For example, we could start with any concave function such as:

$$\hat{Q}^0(R) = \rho_0 \left(1 - e^{-\rho_1 R}\right)$$
$$\hat{Q}^0(R) = \ln(R+1)$$
$$\hat{Q}^0(R) = -\rho_0 (R - \rho_1)^2$$

where R is a scalar. As an alternative, we could initialize the function by assuming independence between the demands. Continuous functions can be converted to piecewise linear functions by extrapolating the function between integer values of R. Let $\tilde{q}^n = q(\omega^n)$ be a stochastic subgradient of Q (given by the dual variable of equation (27)), and let R^n be the resource vector at the n^{th} iteration. We can then update our approximation $\hat{Q}$ using the following updating equation:

$$\hat{Q}^{n+1}(R) = \hat{Q}^n(R) + \alpha^n \left(\tilde{q}^n - \nabla \hat{Q}^n(R^n)\right) \cdot R \tag{31}$$

This strategy, dubbed the "SHAPE" algorithm, is provably convergent when the function $Q(R)$ (and its approximations $\hat{Q}^n(R)$) are continuously differentiable, but in trans-

portation, we are typically managing discrete resources, and we are interested in integer solutions.

When we are using piecewise linear functions, we can get an even better estimate by using left and right gradients of $Q(R_0^c, R_1^o(\omega))$ rather than a simple subgradient. Let $\tilde{q}^{n+}$ and $\tilde{q}^{n-}$ be the right and left gradients, respectively, of $Q(R_0^c, R_1^o(\omega))$. Then we can perform a two-sided update using:

$$\hat{Q}^{n+1}(R) = \begin{cases} \hat{Q}^n(R) + \alpha^n \left(\tilde{q}^{n+} - \nabla \hat{Q}^n(R^n)\right) \cdot R & R \geq R^n \\ \hat{Q}^n(R) + \alpha^n \left(\tilde{q}^{n-} - \nabla \hat{Q}^n(R^n)\right) \cdot R & R < R^n \end{cases} \tag{32}$$

There is another class of strategies that we refer to broadly as structured adaptive functional estimators (or "SAFE" algorithms). In our problem, we are trying to estimate piecewise linear, concave functions which can be represented by a sequence of slopes that are decreasing monotonically. At each iteration, we obtain stochastic gradients that allow us to update estimates of these slopes, but it is important to maintain the concavity of our function or, equivalently, the monotonicity of the slopes. We briefly review two strategies for performing this estimation. The first is referred to as a *leveling* technique since violations of concavity are fixed by leveling the estimates of the slopes (see below). The second is called a separable, projective approximation routine (SPAR), since we maintain monotonicity in the slopes by performing a projection of the updated function onto the space of concave functions.

Both approaches begin by representing the piecewise linear function $Q(R)$ by its slopes as follows. Let:

$$q_r = Q(r+1) - Q(r)$$

be the right derivative of $Q(R)$ at $R = r$. We can then write:

$$Q(R) = Q(0) + \sum_{r=0}^{R-1} q_r$$

Let $\hat{q}_r^n$ be an estimate of q_r at iteration n. As before, let $\tilde{q}^n$ be a stochastic gradient of Q at iteration n, and assume they have the property that $E[\tilde{q}_r^n] = q_r$. Assume that at iteration n we sample $r = R^n(\omega)$. We could estimate the slopes using the simple updating equations:

$$\hat{q}_r^{n+1} = \begin{cases} (1-\alpha^n)\hat{q}^n + \alpha^n \tilde{q}^n & \text{if } r = R^n(\omega) \\ q_r^n & \text{Otherwise} \end{cases} \tag{33}$$

If we assume assume that we are going to sample all the slopes infinitely often, then it is not hard to show that $\lim_{n\to\infty} \hat{q}_r^n = q_r$. But this updating scheme would not work in practice since it does not maintain the concavity of the function $\hat{Q}^n(R)$. We know from the concavity of $Q(R)$ that $q_0 \geq q_1 \geq \ldots \geq q_r$. It is apparent that equation (33) would not maintain this relationship between the slopes. Within an algorithm, this forces us to solve nonconcave optimization problems, which is quite hard. We note that concavity is automatically maintained in equation (31) since we are updating a concave approximation with a linear updating term. Concavity is also maintained in equation (32), since we are guaranteed that $\tilde{q}^{n-} \geq \tilde{q}^{n+}$.

The first of our two approaches maintains concavity (monotonicity in the slopes) by using a technique that we call *leveling*. Here, all we are doing is identifying a violation of concavity after a basic update (as in equation (33)), and then adjusting the neighbors of the updated slope so that concavity is maintained. As before, let $R^n(\omega)$ be the point that we sample in iteration n. The updating equations are given by:

$$\hat{q}_r^{n+1} = \begin{cases} \alpha^n \tilde{q}^n(\omega) + (1-\alpha^n)\hat{q}_r^n & \text{if } R^n(\omega) = r \\ \alpha^n \tilde{q}^n(\omega) + (1-\alpha^n)\hat{q}_i^n & \text{if } R^n(\omega) = i < r \text{ and } \alpha^n \tilde{q}^n(\omega) + (1-\alpha^n)\hat{q}_i^n < \hat{q}_r^n \\ & \\ \alpha^n \tilde{q}^n(\omega) + (1-\alpha^n)\hat{q}_i^n & \text{if } R^n(\omega) = i > r \text{ and } \alpha^n \tilde{q}^n(\omega) + (1-\alpha^n)\hat{q}_i^n > \hat{q}_r^n \\ \hat{q}_r^n & \text{otherwise} \end{cases} \tag{34}$$

The second method starts with the estimate of the slopes given by equation (33) and then performs a projection onto the space of functions whose slopes that are monotonically decreasing. We start by letting the left hand side of (33) be denoted by the vector $\bar{q}^{n+1}$ which clearly may violate concavity. We can now think of $\mathcal{Q}$ as the space of concave functions, and let $\Pi_{\mathcal{Q}}$ be the nearest point projection onto the space $\mathcal{Q}$. This allows us to represent the process of converting the vector $\bar{q}^{n+1}$ as the projection:

$$\hat{q}^{n+1} = \Pi_{\mathcal{Q}}(\bar{q}^{n+1}) \tag{35}$$

The projection $\Pi_{\mathcal{Q}}$ is the solution to the quadratic programming problem:

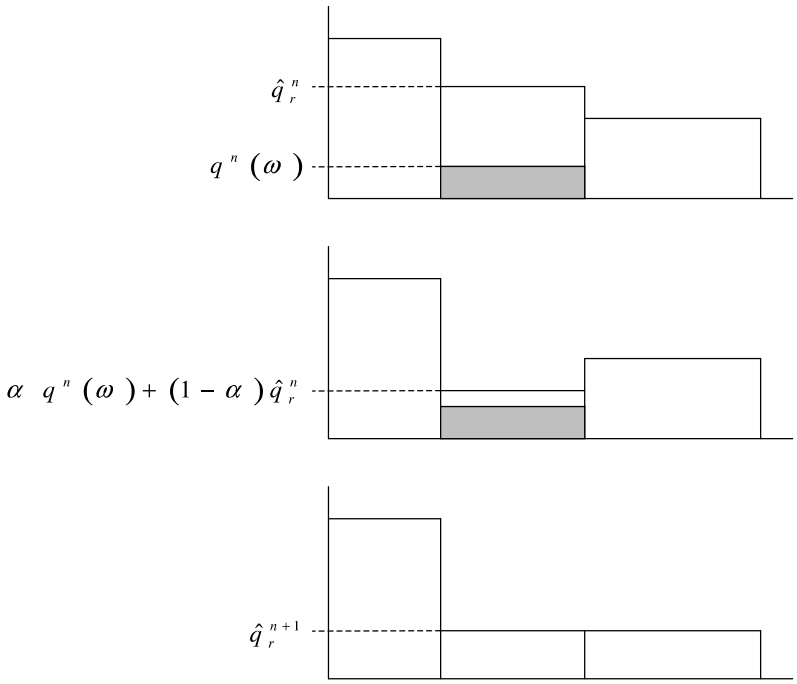

Fig. 6. Maintaining concavity by the leveling method

$$\min_q \|q - \bar{q}^n\|^2 \tag{36}$$

subject to:

$$q_{r+1} - q_r \leq 0 \tag{37}$$

Solving this projection problem is very easy. Assume that after the basic update, we have an instance where $\bar{q}^n_{r-1} < \bar{q}^n_r$. Let $\bar{r} = \arg\min_{r'<r}\{\bar{q}^n_{r'} < \bar{q}^n_r\}$ be the smallest index such that $\bar{q}^n_{\bar{r}} < \bar{q}^n_r$. Now find the average over all these elements:

$$\bar{q}^n_{[\bar{r},r]} = \frac{1}{r-\bar{r}+1}\sum_{r'=\bar{r}}^{r} \bar{q}^n_{r'}$$

Finally, we let

$$\hat{q}^{n+1}_{r'} = \begin{cases} \bar{q}^n_{[\bar{r},r]} \text{ if } \bar{r} \geq r' \geq r \\ \bar{q}^n_r \quad \text{Otherwise} \end{cases}$$

Both the leveling method and the projection method produce convergent algorithms from two perspectives. First, if all the slopes are sampled infinitely often, then we obtain that $\lim_{n\to\infty} \hat{q}_r^n = q_r$ for all r a.s. But, we are not going to sample all the slopes infinitely often. What we want to do is to use the approximation $\hat{Q}^n$ as an approximation of the second stage to determine the solution to the first stage. Thus, our algorithm is going to proceed by solving:

$$x_0^n = \arg\max_{x_0} c_0 x_0 + \hat{Q}^n(R_0(x_0)) \tag{38}$$

subject to our first stage constraints:

$$\sum_{d\in\mathcal{D}} x_{0ad} = R_{0,a}^{+,c} \quad a \in \mathcal{A} \tag{39}$$

$$\sum_{a\in\mathcal{A}} x_{0ad}\delta_{1,a'}(0,a,d) = R_{0,a'}^{c} \quad a' \in \mathcal{A} \tag{40}$$

$$\sum_{a\in\mathcal{A}} x_{0ad} \leq R_{0,a_d}^{+,o} \quad d \in \mathcal{D}^o \tag{41}$$

$$x_{0ad} \in Z_+ \tag{42}$$

The problem (38)-(42) is a pure network identical to that shown in figure 5. Once we obtain x_0^n, we find a sample realization ω^n and solve the optimization problem in (26) again. The duals from this problem are used to update the value function, and the process repeats itself.

Our algorithm, then, does not sample the entire domain for R_0, but rather only those that are produced by solving our first stage approximation. Fortunately we can show that this algorithm will visit the optimal solution infinitely often. A side benefit is that we are solving sequences of pure networks which readily yield integer solutions. Integrality can be a major headache in transportation applications, but we have now designed an algorithm which always produces integer solutions. Of central importance in this regard is the fact that our algorithm never performs smoothing on the decision variables, as would be required if we used stochastic linearization methods.

At this point it may seem that we are simply solving very special cases. In fact, as we soon show, we actually have all the machinery we need to solve very general instances of this problem.

The next level of generalization is the challenge of working with what we call "classification yards." For the purpose of this presentation, we are going to assume that we can send cars from classification yards to any customers in the network (for the moment, we are not going to allow ourselves to send cars to regional depots, since this would take us past our basic two-stage model). For the moment, we are going to continue to assume that once cars reach a classification yard that there is no substitution between car types: if a customer order is for car type k, then we must provide car type k. But we are going to assume that a single customer can be served from more than one depot.

This problem is known to the stochastic programming community as a two-stage stochastic program with network recourse. The reason is that, unlike our previous models, the second stage is now a general network problem (as opposed to the much simpler problems posed by simple or nodal recourse). Solving a network problem in the second stage is almost as difficult as solving a general linear program, which means that we should consider algorithms designed for general two-stage stochastic linear programs.

The research community has developed a number of algorithmic strategies over the years. The question of whether an algorithmic strategy works has to be answered in three levels: 1) Does the algorithm appear to work in theory? Does it capture the mathematical properties of the problem? 2) Does it produce reasonable numerical results in laboratory experiments? For example, using datasets reflecting specific classes of problems, we would like to know if it converges quickly, producing stable, high quality solutions. 3) Does it work in practice, producing recommendations that are acceptable to those who have to implement them?

As of the writing of this chapter, there is a strong handle on the theory, but numerical testing is extremely limited (given the broad diversity of problems). For example, showing that it works well on car distribution problems for one railroad will not always convince another railroad that it will work on their network! Container management problems (in trucking, rail and intermodal applications) come in a variety of sizes and characteristics. The dynamics of short-haul regional truckload carriers are completely different from those of long-haul national carriers. Experiments in pure transportation applications do not tell us whether it would work in other resource allocation settings such as supply chain management and distribution problems. And we are not even talking about applications outside of transportation and logistics. In short, each subcommunity (and these can be very specialized) needs to see numerical work to demonstrate effectiveness on its own problem class.

Given the very limited amount of laboratory testing of the different algorithmic strategies (even within general transportation problems), our discussion focuses on the qualities of

different algorithms and their potential strengths and weaknesses for our problem. We cannot definitively state what will and will not work for our problem class, but we can discuss the qualities of different approaches. In particular, we are interested in the degree to which a technique allows us to exploit the underlying structure of the transportation problem. Many transportation problems require integer solutions, and also exhibit near-network structure. Algorithms which allow us to exploit this network structure are more likely to yield integer solutions from LP relaxations, or at least provide tight LP relaxations.

5.6.1 Scenario methods

Perhaps the best known algorithmic strategy in stochastic programming is *scenario programming*, popular because of its conceptual simplicity, generality, and use of general-purpose optimization algorithms. But, its effectiveness for transportation applications is doubtful.

Let $\hat{\Omega}$ be a (not too large) sample of outcomes (future car orders, future car supplies, as well as travel times). Further let $\hat{p}(\omega)$ be the probability of outcome $\omega \in \hat{\Omega}$. We can approximate our original problem using the method of scenarios:

$$\max_{x_0, x_1} c_0 x_0 + \sum_{\omega \in \hat{\Omega}} \hat{p}(\omega) c_1 x_1(\omega) \tag{43}$$

subject to:

First-stage constraints:

$$\sum_{d \in \mathcal{D}} x_{0ad} = R^c_{0a} \quad a \in \mathcal{A} \tag{44}$$

$$\sum_{a \in \mathcal{A}} x_{0ad} \leq R^o_{0a_d} \quad d \in \mathcal{D}^o \tag{45}$$

$$\sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}_a} x_{0ad} \delta_{1,a'}(0, a, d) - R^c_{0,a'1} = 0 \quad a' \in \mathcal{A} \tag{46}$$

$$x_{0ad} \in Z_+ \quad a \in \mathcal{A}, d \in \mathcal{D} \tag{47}$$

Second-stage constraints:

$$\sum_{d \in \mathcal{D}_a^c} x_{1ad}(\omega) + x_{1ad^\phi}(\omega) = R_{0,a}^c \quad \forall a \in \mathcal{A}, \quad \forall \omega \in \hat{\Omega} \tag{48}$$

$$\sum_{a \in \mathcal{A}} x_{1ad}(\omega) \leq R_{1c_d}^o(\omega) \quad d \in \mathcal{D}^c, \quad \forall \omega \in \hat{\Omega} \tag{49}$$

$$x_{1ad}(\omega) \geq 0 \quad \forall a \in \mathcal{A}, d \in \mathcal{D}_a^c, \quad \forall \omega \in \hat{\Omega} \tag{50}$$

We note that our decision class $\mathcal{D}$ only allows us to assign cars to a known order, or to reposition cars to a general or regional depot. As a result, the second stage problem is primarily one of assigning cars from the general and regional depots to orders that became known in the second stage.

Scenario methods have been very popular in financial applications, but we feel that there are specific characteristics of financial applications that are not shared in transportation applications, and vice versa. Financial applications are characterized by very complex stochastic processes with high levels of interdependence reflecting the dependence of random outcomes on a relatively smaller number of common factors. It is easier, then, to reasonably approximate the future with a smaller number of scenarios. Also, financial applications typically do not have integer variables.

Transportation applications, on the other hand, are characterized by a large number of relatively independent random variables. The optimization problems, which are typically integer, are often so large that deterministic problems are hard (although they often have imbedded network structures). The formulation in (43)-(50) has the effect of taking a computationally intractable problem and blowing it up into a problem that is many times larger. Furthermore, many transportation problems exhibit a natural network structure that is destroyed by the representation of the second stage problem.

5.6.2 Benders decomposition

Benders decomposition is an appealing algorithm that replaces the very large problems posed in scenario optimization with sequences of relatively small problems of the form (which we state here as a minimization problem as is standard practice in the literature):

$$\min_{x_0} \quad c_0 x_0 + z \tag{51}$$

subject to the first stage constraints (44)-(47) which we represent compactly using:

$$A_0 x_0 = R_0 \tag{52}$$

$$x_0 \geq 0 \tag{53}$$

and the constraints:

$$z - \beta_i x_0 \geq \alpha_i, \quad \forall \quad i = 1, ..., n \tag{54}$$

where β_i and α_i are generated by solving the dual of the second stage problem, which for compactness we can write as:

$$\min_{x_1} c_1 x_1(\omega)$$

subject to:

$$\begin{aligned} A_1 x_1(\omega) &= R_1(\omega) + B_0 x_0 \\ x_1(\omega) &\geq 0 \end{aligned}$$

Different algorithms have been proposed for generating cuts. The first algorithm of this class is the so-called "L-shaped" decomposition, which works on a finite set of outcomes (which cannot be too large, since we have to solve a linear program for each outcome). This concept was generalized by the stochastic decomposition algorithm which generates cuts from a potentially infinite sample space. A sketch of the algorithm is given in figure 7. This algorithm converges almost surely to the optimal solution, but the rate of convergence on practical applications remains an open question.

The CUPPS algorithm (outlined in figure 8) requires a finite sample space which can be quite large (for example, thousands or tens of thousands of scenarios). The critical step in the stochastic decomposition is equation (56) which requires smoothing on the coefficients of the cuts. The critical step in the CUPPS algorithm is equation (57) which requires a simple arithmetic calculation over the entire sample space. Since equation (57) is quite simple, it is not hard to execute even for tens of thousands of scenarios, but it prevents the algorithm from ever being applied rigorously to complete sample spaces (for realistic problems) which can be of the order 10^{10} or even 10^{100}. From a practical perspective, it is not clear if this is useful.

We need to keep in mind that Benders decomposition is probably limited (in transportation applications) to the types of resource allocation problems that we have been considering (since these can be reasonably approximated as continuous linear programs). However, there are unanswered experimental questions even for this special problem class. First, there is the usual issue of rate of convergence. Real car distribution problems may have over 100 regional depots and thousands of customers (for our model, it is the number of regional depots that really impacts the second stage problem). If there are 50 car types (a conservative estimate) and 100 depots, then (realizing that we do not have all car types at

Step 1. Solve the following *master problem*:

$$x_0^n = \arg\min\{c_0 x_0 + z : A_0 x_0 = R_0,\ z - \beta_t^n x \geq \alpha_t^n,\ t = 1, \ldots, n-1,\ x \geq 0\}$$

Step 2. Sample $\omega^n \in \Omega$ and solve the following *subproblem*:

$$\min\{c_1 x_1 : A_1 x_1 = R_1(\omega^n) + B_0 x_0^n,\ x_1 \geq 0\}$$

to obtain the optimal dual solution:

$$v(x_0^n, \omega^n) = \arg\min_v\{(R_1(\omega^n) + B_0 x_0^n)v : A_1^T v \leq c_1\}$$

Augment the set of dual vertices by:

$$\mathcal{V}^n = \mathcal{V}^{n-1} \bigcup \{v(x_0^n, \omega^n)\}$$

Step 3. Set:

$$v_t^n = \arg\max\{(R_1(\omega^t) + B_0 x_0^n)v : v \in \mathcal{V}^n\} \text{ for all } t = 1, \ldots, n$$

Step 4. Construct the coefficients of the n^{th} cut to be added to the master problem by:

$$\alpha_n^n + \beta_n^n x_0 \equiv \frac{1}{n} \sum_{k=1}^{n} \Big(R_1(\omega^k) + B_0 x_0\Big) v_k^n \tag{55}$$

Step 5. Update the previously generated cuts by:

$$\alpha_k^n = \frac{n-1}{n} \alpha_k^{n-1}, \quad \beta_k^n = \frac{n-1}{n} \beta_k^{n-1}, \quad k = 1, \ldots, n-1 \tag{56}$$

Fig. 7. Sketch of the stochastic decomposition algorithm

all locations) we can still anticipate upwards of a thousand resource states for the second stage problem. How quickly does Benders decomposition converge for problems of this size? The problem is that a single cut may not improve our approximation of the value of cars in a particular location.

A second issue with Benders decomposition is that real applications require integer solutions. When the flows are relatively large, solutions are easily rounded to obtain reasonable

Step 1. Solve the following *master problem*:

$$x_0^n = \arg\min\{cx + z : A_0 x_0 = R_0,\ z - \beta_k^n x \geq \alpha_k^n,\ k = 1, \ldots, n-1,\ x \geq 0\}$$

Step 2. Sample $\omega^n \in \Omega$ and solve the following dual *subproblem*:

$$v(x^n, \omega^n) = \arg\min\{(R^o(\omega^n) + B_0 x_0^n)v : A_1^T v \leq c_1\}$$

Augment the set of dual vertices by:

$$\mathcal{V}^n = \mathcal{V}^{n-1} \bigcup \{v(x^n, \omega^n)\}$$

Step 3. Set:

$$v^n(\omega) = \arg\max\{(R^o(\omega) + B_0 x_0^n)v : v \in \mathcal{V}^n\} \text{ for all } \omega \in \Omega \tag{57}$$

Step 4. Construct the coefficients of the n^{th} cut to be added to the master problem by:

$$\alpha_n^n + \beta_n^n x_0 \equiv \sum_{\omega \in \Omega} p(\omega)\,(R_1(\omega) + B_0 x_0)\,v^n(\omega)$$

Fig. 8. Sketch of the CUPPS algorithm

approximations to the discrete version of the problem. In actual applications, there can be many instances where flows (to small locations, or of unusual car types) are quite sparse and fractional solutions become more problematic. In these cases, fractional solutions can involve flows that are less than one, and simple rounding may produce infeasible solutions. At a minimum, dealing with fractional solutions can be a nuisance.

Despite these questions, Benders decomposition is a promising technique that needs to be tested in the context of specific applications.

5.6.3 Stochastic linearization techniques

A powerful set of techniques is based on linear approximations of the recourse function. These fall into two groups. The first uses pure linear approximations, but performs smoothing on the first stage variables to stabilize the solution. The second group introduces some sort of nonlinear stabilization term.

The pure linearization strategy solves sequences of problems of the form:

$$\hat{x}_0^n = \arg\min_{x_0} c_0 x_0 + \bar{q}^{n-1} \cdot R_1(x_0) \tag{58}$$

To calculate $\bar{q}^n$, let $\tilde{q}^n$ as before be our stochastic gradient obtained as the dual variable of the resource constraint from the second stage using a Monte Carlo realization ω^n. We then smooth these gradients to obtain:

$$\bar{q}^n = (1 - \alpha^n)\bar{q}^{n-1} + \alpha^n \tilde{q}^n \tag{59}$$

Having obtained the solution $\hat{x}_1^n$, we then smooth this as well:

$$\bar{x}_1^n = (1 - \beta^n)\bar{x}_1^{n-1} + \beta^n \hat{x}_1^n \tag{60}$$

For both equations (59) and (60), we would normally require the usual conditions on the stepsizes for stochastic problems, namely that $\sum_{n=0}^{\infty} \alpha^n = \infty$ and $\sum_{n=0}^{\infty} (\alpha^n)^2 < \infty$, although we note in passing that this is not normally satisfied by the stepsize rules used in practice.

Stochastic linearization techniques are clearly the simplest to use, but are unlikely to work in practice simply because of the lack of stability. Stability is imposed on the solution primarily through the use of declining stepsizes, but this is an artificial form of stability. Furthermore, the smoothing on the first stage variables performed in (60) is a serious practical problem because it completely destroys integrality. Rounding is hard for large problems because it can be difficult ensuring conservation of flow in the presence of substitutable resources.

Despite these weaknesses, techniques based on linear approximations are attractive for transportation applications since they retain the structure of the original problem. If the first stage is a network problem, then adding a linear adjustment term retains this property. Linear approximations are also easy to compute and store. We may be able to overcome the instability of pure linearization techniques by employing any of a variety of nonlinear stabilization terms. One notable example is proximal point algorithms, which solve sequences of problems of the form:

$$x_0^n = \arg\min_{x_0} c_0 x_0 + \bar{q}^{n-1} \cdot R_1(x_0) + \theta\psi(x_0, \bar{x}_0^{n-1})$$

where $\psi(x, \bar{x}^{n-1})$ is a distance metric such as $\psi(x, \bar{x}^{n-1}) = \|x - \bar{x}^{n-1}\|^2$. $\bar{x}^n$ is computed using:

$$\bar{x}^n = (1 - \beta^n)\bar{x}^{n-1} + \beta^n x^n$$

Note that at the last iteration, the final solution is x^n, not $\bar{x}^n$. $\bar{x}^n$ is used only to stabilize the solution.

In engineering practice, we can be creative in the construction of the distance metric $\psi(x, \bar{x}^{n-1})$. If it is separable in x, then we may find ourselves solving a separable, convex optimization problem. If we are particularly interested in integer solutions, we can construct a piecewise linear function defined on discrete values of x (even if $\bar{x}^{n-1}$ is fractional).

A second type of nonlinear stabilization strategy is the SHAPE algorithm first presented in section 5.5.2. This is a type of auxiliary function algorithm where we start with an artificial auxiliary function $\hat{Q}^0(R)$, and then update it using stochastic gradients as demonstrated in equation (31). An attractive feature of this algorithm is that the auxiliary function can (and should) be chosen to retain the structure of the first stage problem as much as possible. For our car distribution problem (and similar applications), piecewise linear, separable approximations are particularly attractive.

5.6.4 Nonlinear functional approximations

Our last class of strategies tries to explicitly approximate the recourse function, without any guarantee of convergence to the exact function. We do not include algorithms such as the one-sided SHAPE algorithm (equation (31)) in this group because there is no explicit attempt to approximate the recourse function. We also do not include Benders decomposition simply because we feel that this strategy is in a class by itself. But we do include both the two-sided SHAPE algorithm (equation (32)) and the structured adaptive functional estimators.

We first introduced these algorithms in the context of a recourse function $Q(R)$ which could be written as a separable function of the resource vector R. Now consider what would happen if we apply the exact same algorithms to a nonseparable problem. We still produce a separable approximation of Q, and we still solve sequences of networks that are identical to figure 5. The important difference is that we are solving sequences of separable approximations of otherwise nonseparable functions. For continuously differentiable problems, this can be an optimal strategy (in the limit). For nondifferentiable problems (as we are considering) the result is an algorithm that is very near optimal with a much faster rate of convergence than has been achieved using Benders decomposition.

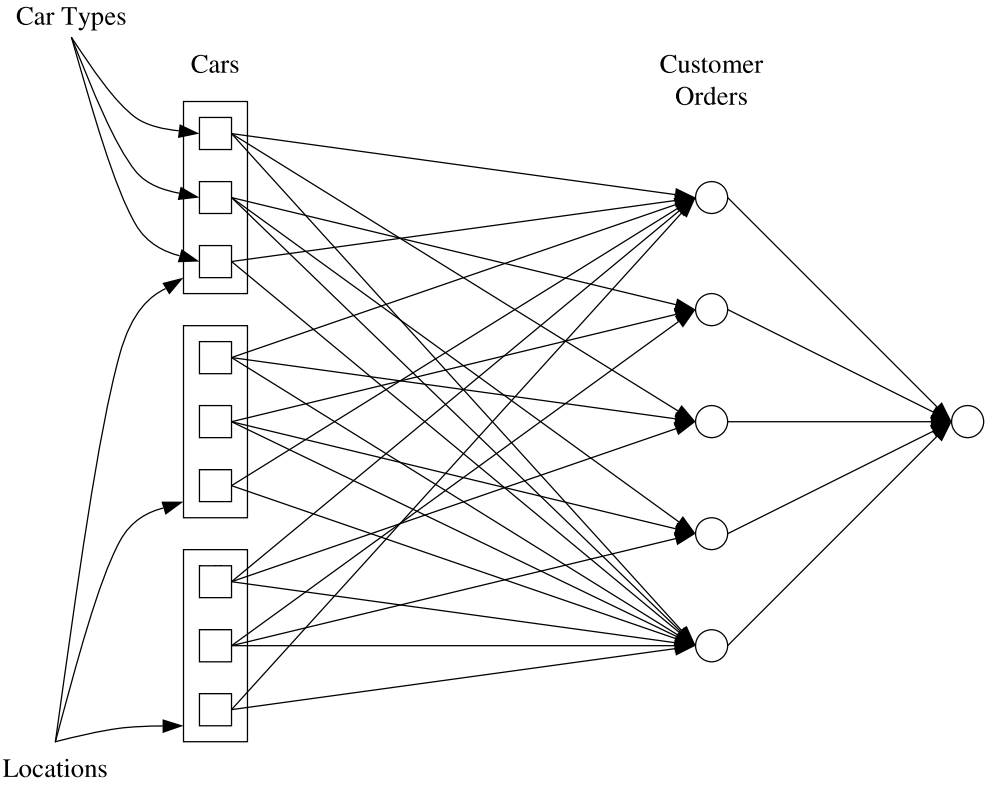

Fig. 9. Illustration of second stage substitution between car types and locations

5.6.5 Extension to substitution across car types

In the previous section, we retained our assumption that there was no substitution between car types in the second stage. However, the substitution between classification yards (spatial substitution) produced a problem with network recourse (we had to solve a transshipment problem in the second stage). Now consider what happens if we allow substitution between car types, which produces the second stage network illustrated in figure 9. We quickly see that while this expands the set of options for a car, it is still a network, and is mathematically equivalent to the problem which only allows spatial substitution. In addition, we are also implicitly allowing temporal substitution. In the second stage, we will forecast demands that are actionable on different days, as well as cars that will become available on different days. However, we are allowing general assignments of cars to demands that may require cars to be held to meet the demand, or require demands to wait until the car arrives.

We see, then, that we can use the same algorithmic strategy for handling substitution between car types as we did for geographic substitution. But this avoids the more practical question: will it work? It is much more convincing to argue that spatial problems will

produce approximately separable recourse functions than would arise in the case with other forms of substitution. For example, it is quite likely that the cost of substituting other box cars in the same car group is quite small. In fact, this is the reason that a railroad might reasonably ignore car subgroups and just model car groups.

For two-stage problems, there is a strong reason to believe that separable approximations will work well, even when the recourse function is not even approximately separable. The logic behind this argument is that for n sufficiently large, $\hat{Q}^n$ will stabilize, and therefore so will x_0^n. As x_0^n (approximately) approaches a limit point, $R_1^n(x_0^n)$ will approach a limit point, allowing us to produce an accurate (but not perfect) piecewise linear approximation $\hat{Q}_j^n(R_j)$ of the j^{th} dimension of $Q(R)$, at the point $R = R_1^n(x_0^n)$. Thus, a separable approximation only needs to be a good local approximation to provide good results.

5.6.6 Summary

The choice of the best algorithm for two-stage resource allocation problems remains an open question. Two-stage problems are an important foundational problem, but in transportation applications, the usual goal is to solve multistage problems. However, it is important to study the performance of an algorithm in a two-stage setting first, and it is difficult to believe that an algorithm that does not work well for two-stage problems would turn out to work well in a multistage application. But the converse may not be true; an algorithm that does work well for a two-stage problem may not work well in a multistage setting. It is possible for an algorithm (such as the separable, nonlinear functional approximations) to exploit the limiting behavior of the first stage decisions, a property that we lose immediately in the context of multistage problems.

Our belief is that while scenario methods are unlikely to prove attractive in practice, the other three major classes of techniques (Benders decomposition, stochastic linearization with nonlinear stabilization strategies, or nonlinear functional approximations) all deserve serious study. We believe that nonlinear functional approximations are going to work the best for two-stage problems because a) they attempt to explicitly approximate the recourse function and b) they exploit the structure of problems that arise in transportation. However, we have not yet addressed some of the more difficult dimensions of transportation applications (some of which are touched on below). Nonlinear approximations can work the best in laboratory experiments, but are much more difficult to use than linear approximations. Pure linear approximations are too unstable, but linear approximations with nonlinear stabilization terms (proximal point algorithms or auxiliary functions) may offer an attractive alternative. All of these approximations are separable, and it is simply not clear how well these will work in multistage applications.

5.7 Extension to large attribute spaces

Up to now, we have considered a problem where there are $|\mathcal{K}|$ car types spread among $|\mathcal{I}|$ locations. Using our earlier notation, the attribute vector of a car would be represented by $a = (k, i)$. Thus, our attribute space $\mathcal{A}$ has $|\mathcal{A}| = |\mathcal{K}| \times |\mathcal{I}|$ elements. A large railroad might have several hundred regional depots and 50 to 100 car types, with a total of several thousand combinations. Large, but not hard to enumerate on modern computers.

Now consider what a real car distribution problem looks like. While there are, in fact, 50 to 100 real car types, these cars are allocated among several dozen *pools* which represent groups of cars controlled by individual shippers. The allocation of cars to pools is negotiated periodically between the shipper and the railroad, and cars can be moved from one pool to another, reflecting the evolving needs of the shippers. Cars are also characterized by the railroad that owns them (when they are not in a pool). Box cars sometimes carry dirty freight that limits their ability to carry cargo (such as food) that requires a clean car. For this reason, cars carry the attribute of the commodity type that they last carried. Finally, some cars may have equipment problems that require maintenance. These problems may range from minor issues that do not affect the use of the car to more serious equipment problems.

When we use this full vector of attributes, we now find that there are not several thousand possible attributes, but several million. Now we cannot even generate the entire attribute space. This creates an interesting problem. In our optimization model, we may wish to consider acting on a car with what is now a multiattribute vector a with decision d, producing a car with attribute vector a'. Since we are not able to enumerate the space $\mathcal{A}$, we may not have an approximation for $\hat{Q}^n_{a'}(R_{a'})$. As a result, we have to devise a strategy for approximationg $\hat{Q}^n_{a'}(R_{a'})$.

We are not able to address this issue in depth, but it is important to understand some of the problems that arise. First, it is easy to see that we should try to make sure that our initial approximation $d\hat{Q}^0_{a'}(R_{a'})/dR_{a'}\Big|_{R_{a'}=0}$ is an optimistic estimate of $\partial Q(R)/\partial R_{a'}|_{R=0}$. If we did not do this, then a low estimate might result in us choosing not to make the decision d that produces a resource with attribute a'. Since we never visit that resource state, we never improve our approximation.

In practice, the use of optimistic estimates of the value of a resource may not work. We have found that initial approximations that are guaranteed to be optimistic are actually too optimistic. Consider choosing between decisions d' and d''. Assume that decision d' produces a resource with attribute a' while decision d'' produces a resource with attribute a''. Further assume that we have generated the attribute a' (and therefore have an approx-

imation $\hat{Q}^n_{a'}(R_{a'})$), but we have never generated the attribute a''. If we use an optimistic approximation for $\hat{Q}^n_{a''}(R_{a''})$, then we would choose d'' just because we have never tried it before. The result is the steady exploration of every possible decision, and a virtual enumeration of the attribute space $\mathcal{A}$.

A more practical approach is to assume that we have access to an aggregation function $G(a) \mapsto \hat{a}$ where $\hat{a} \in \hat{\mathcal{A}}$ is an aggregation of the original attribute space. We assume that $\hat{\mathcal{A}}$ is not too large and can be enumerated. We further assume that $\hat{Q}^n_{\hat{a}}(R_{\hat{a}})$ is a "good" approximation of $\hat{Q}^n_a(R_a)$ when $G(a) = \hat{a}$. We then make sure that we repeatedly sample gradients and update $\hat{Q}^n_{\hat{a}}(R_{\hat{a}})$ for all $\hat{a} \in \hat{\mathcal{A}}$.

We are not aware of any formal convergence theory for this approach (or for any other algorithm for this problem class). But real problems in transportation are characterized by a much richer vector of attributes than is normally considered by the academic community.

6 Multistage resource allocation problems

We now turn to the challenge of solving multistage problems. In multistage problems, we have to capture the sequential decision-making process as information (and decisions) evolve over time. For transportation problems, we encounter the reusability of resources; once a vehicle moves a load, it is available to be used again.

We could motivate our multistage problem using our rail car distribution example, but it is useful to bring other applications into the picture. Examples include:

1) Fleet management for truckload trucking - In truckload trucking, a truck moves an entire load of freight from origin to destination. Uncertainty plays a major role in the long haul truckload market, where loads can take between one and four days to deliver. Customers sometimes request trucks the same day the order is made, but more often call in one or two days in advance. In sharp contrast with the rail industry, the truckload carrier does not have to accept every load, and this is one of the major challenges. Emerging electronic market places, where loads are posted on web sites, open the possibility of taking loads that do not have to move for several days. This is a classic problem of decision-making under uncertainty.
2) Driver management for long-haul less-than-truckload motor carriers - LTL carriers face the problem of timing the movement of loads over the network, requiring the careful management of drivers.
3) Management of jets in the fractional ownership industry - In this business, high net worth individuals and business executives will own a fraction of a jet. This gives them

access to the entire fleet of jets. They may call the company with as little as eight hours notice and request that a jet move them from a local airport to any other airport. After the move, the fleet operator will move the jet to another location.

4) Routing and scheduling transport aircraft for the air mobility command - The AMC works like a large trucking company, moving freight for the military using large transport aircraft. They are typically used in support of emergency situations where requests for freight movement arise dynamically.

Compared to our rail car distribution problem, these applications are characterized by relatively shorter travel times and less flexibility to satisfy customer orders at times other than when they were requested.

Multistage problems are, of course, much harder than two-stage problems, but we are going to approach them by building on the tools we have already introduced. Our strategy for solving multistage problems is to solve them as sequences of two-stage problems. Students of dynamic programming will see strong similarities in our modeling approach. But multistage problems do introduce a fresh set of modeling and algorithmic issues that simply do not arise in two-stage problems.

We start in section 6.1 with a formulation of the problem. Then, Section 6.2 outlines the general algorithmic strategy. Section 6.3 describes the implementation in the context of single commodity flow problems. Section 6.4 outlines the challenges that arise when we solve multicommodity problems in a multistage setting. Finally, section 6.5 describes the complications that are introduced when we model the property that it takes more than one time period to go from one location to another.

6.1 Formulation

We present the basic multistage problem with somewhat more generality than we have used previously. We first define the exogenous information arriving to our system:

$\hat{R}_t =$ Vector of new arrivals in period t, where $\hat{R}_t = (\hat{R}^o_t, \hat{R}^c_t)$.

$\xi_t =$ Complete vector of new information arriving in period t, including both $\hat{R}_t$ as well as other information about system parameters (travel times, costs, and parameters governing the physics of the problem).

For our purposes, we are only interested in the resource state R_t, and the only information process we are modeling at the moment is the arrival of new resources, $\hat{R}_t$. Using this notation, our history of states, information and decisions (given earlier in equation (7)) would look like:

$$h_t = \{R_0^+, x_0, R_0, \hat{R}_1, R_1^+, x_1, R_1, \hat{R}_2, R_2^+, x_2, R_2, \ldots, \hat{R}_t, R_t^+, x_t, R_t, \ldots\} \tag{61}$$

There are three perspectives of the state of our system:

$R_t =$ Vector of resources available in period t after decisions x_t have been made.
$K_t =$ What is known at time t after the new information ξ_t has been incorporated. K_t includes R_t plus what we know about parameters that govern the dynamics of the system.
$I_t =$ Set of information available at time t for making a decision. I_t includes K_t, but it might also include forecasts of future activities (activities which are not "known" now, but are the result of a forecasting exercise).

Our process is controlled by the decisions we make:

$x_t =$ Decisions which are made after new information in period t has become known.
$X_t^\pi(I_t) =$ The decision function of policy π.

Our decisions are chosen to maximize the expected total contribution over a planning horizon. Our contribution function is expressed as:

$c_t(x_t, K_t) =$ The contribution generated in period t given decision x_t, and what is known, K_t.

When resources are allocated in time t, they have to satisfy flow conservation equations of the form:

$$\sum_{d \in \mathcal{D}} x_{tad} = R_{t-1,at} + \hat{R}_{t,at}$$

where we assume that we can only act on resources that are actionable now (R_{tt}). The physical dynamics of the system are given by:

$$R_{t,a't'} = R_{t-1,a't'} + \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}_a} \delta_{t',a'}(t,a,d) x_{tad} \quad \forall a' \in \mathcal{A}, \quad t' \geq t+1. \tag{62}$$

It is often useful to express flow conservation and system dynamics in matrix form, which we do using:

$$A_t x_t = R_{t-1} + \hat{R}_t \tag{63}$$
$$R_t - B_t x_t = 0 \tag{64}$$

For reasons that are made clear in section 6.2, we have written our equations directly in terms of the incomplete resource vector R_t. The complete resource vector is simply $R_t^+ = R_{t-1} + \hat{R}_t$.

The informational dynamics can be written generally as:

$$K_{t+1} = U^K(K_t, \xi_{t+1}) \tag{65}$$

which is how we would represent the process of updating demand forecasting equations, parameter estimation equations, and the storage of other types of information.

The basic statement of the multistage problem is now given by:

$$\max_{\pi \in \Pi} E \left\{ \sum_{t \in \mathcal{T}} c_t(X^\pi(I_t), K_t) \right\}$$

subject to flow conservation (equation (63)), resource dynamics (equation (64)) and informational dynamics (equation (65)). There may also be upper bounds on flows representing physical constraints.

The challenge, now, is choosing a function $X^\pi(I_t)$. Popular choices include myopic policies ($I_t = K_t$), or rolling horizon procedures, where $I_t = (K_t, \hat{\Omega}_t)$ where $\hat{\Omega}_t$ represents a forecast of future events made with the information known at time t. If $|\Omega_t| = 1$ then we are using a point estimate of the future and we obtain classical deterministic methods for handling the future. In the next section, we discuss how adaptive dynamic programming methods can be used.

6.2 Our algorithmic strategy

Our strategy for solving multistage problems is based on techniques from approximate dynamic programming. Since this approach is not familiar to the stochastic programming community, some background presentation is useful. Recall from section 6.1 (and in particular equation (61) that we can measure the state of our system before or after we make a decision. It is common in the dynamic programming and control community to write the optimality equations using the state before we make a decision, producing optimality equations of the form:

$$Q_t^+(R_t^+) = \arg\max_{x_t} c_t(X^\pi(I_t), R_t^+) + E\left\{ Q_{t+1}^+(R_{t+1}^+) | R_t^+ \right\} \tag{66}$$

Classical dynamic programming techniques are computationally intractable for this problem class. Solving equation (66) using classical discrete dynamic programming techniques encounters three "curses of dimensionality": the state space, the outcome space and the action space. Each of these variables are vectors (and potentially vectors of high dimensionality). Computing the value functions using a backward recursion requires computing equation (66) for each possible value of the state variable. Computing the expectation requires summing over all the outcomes in the outcome space. Finally, since the expected value function may not have any special structure, solving the optimization problem requires evaluating all possible values of the decision vector x_t.

We overcome this problem using the following strategy. First, recognizing that we do not know Q^+_{t+1}, we replace it with an appropriate approximation that for the moment we denote by $\hat{Q}^+_{t+1}(R^+_{t+1})$. Next, we recognize that we cannot compute the expectation in (66). The common strategy is to replace the expectation with an approximation based on a sample taking from the outcome space:

$$Q^+_t(R^+_t) = \arg\max_{x_t} c_t(X^\pi(I_t), R^+_t) + \sum_{\hat{\omega}\in\hat{\Omega}} p(\hat{\omega}) Q^+_{t+1}(R^+_{t+1}(\hat{\omega})) \tag{67}$$

Equation (67) can be exceptionally difficult to solve for the types of high dimensional, discrete resource allocation problems that arise in transportation. It is particularly inelegant when we realize that it is often the case that the myopic problem (maximizing $c_t x_t$) is a pure network, which means the introduction of the approximate value function is making a trivial integer program quite difficult (the LP relaxation is not a good approximation).

Equation (67) is quite easy to solve if we use a linear approximation for $\hat{Q}_{t+1}$. In this case:

$$\begin{aligned}\hat{Q}^+_{t+1}(R^+_{t+1}) &= \hat{q}^+_{t+1} \cdot R^+_{t+1} \\ &= \hat{q}^+_{t+1} \cdot (R_t + A_t x_t + \hat{R}_{t+1})\end{aligned} \tag{68}$$

Taking conditional expectations of both sides of (68) gives:

$$\begin{aligned}E\{\hat{Q}^+_{t+1}(R^+_{t+1})|R^+_t\} &= E\{\hat{q}^+_{t+1} \cdot (R_t + A_t x_t + \hat{R}_{t+1})|R^+_t\} \\ &= \hat{q}^+_{t+1} R_t + \hat{q}^+_{t+1} A_t x_t + E\{\hat{q}^+_{t+1}\hat{R}_{t+1}|R^+_t\}\end{aligned} \tag{69}$$

The only term on the right hand side of (69) involving the expectation is not a function of x_t, so it is only a constant term and can be dropped. The resulting optimization problem is identical to the original myopic optimization problem with a linear adjustment term which would never destroy any nice structural properties of the original problem (for example, network structure).

So, a linear approximation allows us to avoid the problem of taking multidimensional expectations. But what if we use a nonlinear approximation? Now the presence of the expectation presents a serious computational complication. We can circumvent the problem by formulating our optimality recursion around the incomplete state variable R_t. This gives us optimality equations of the form:

$$Q_{t-1}(R_{t-1}) = E\left\{\arg\max_{x_t} c_t x_t + Q_t(R_t(x_t))|R_{t-1}\right\} \tag{70}$$

Again, we propose to replace the recourse function $Q_t(R_t)$ with an approximation $\hat{Q}_t(R_t)$. Note the shift in the time index, which reflects the information content of each variable. We still face the problem of computing (at least approximately) the expectation. However, we are not really interested in computing the expectation. Instead, we need an action x_t which depends on both R_{t-1} and the new arrivals $\hat{R}_t$. For this reason, we would make a decision contingent on a single sample realization, allowing us to write our decision function using:

$$X_t^\pi(R_{t-1}, \hat{R}_t(\omega)) = \arg\max_{x_t} c_t x_t + \hat{Q}_t(R_t(x_t, \omega)) \tag{71}$$

subject to:

$$\sum_{d\in\mathcal{D}} x_{tad} = R_{ta}^c + \hat{R}_{ta}^{c+}(\omega) \quad a \in \mathcal{A} \tag{72}$$

$$\sum_{a\in\mathcal{A}} x_{tad} \leq R_{ta_d}^o + \hat{R}_{ta_d}^{o+}(\omega) \quad d \in \mathcal{D}^o \tag{73}$$

$$x_{tad} \in Z_+ \tag{74}$$

X_t^π is an approximate decision function which can be viewed as a class of policy (in the language of Markov decision theory), where the policy is determined by the choice of $\hat{Q}_t$. Because we have formulated the recourse function in terms of the incomplete state variable, there is no need to directly approximate the expectation (this is being done indirectly through the estimation of $\hat{Q}_t$). We now face the challenge of determining $\hat{Q}_t$. Fortunately, we only have to use the techniques that we described earlier for the two stage problems. Linear approximations remain the simplest to use, but current experimental evidence suggests that piecewise linear, separable approximations are both relatively easy to solve and also provide much higher quality solutions.

Our overall algorithmic strategy is shown in figure 10. We refer to this as the "single-pass" version of the algorithm. We initialize $\hat{Q}_t$ for all t. We then simulate forward in time, using dual variables to update the approximation of $\hat{Q}_t$, sampling new information as the algorithm progresses.

STEP 0: Initialization:
Initialize $\hat{Q}_t^0, \quad t \in \mathcal{T}$.
Set $n = 0$.
STEP 1: Do while $n \leq N$:
Choose $\omega^n \in \Omega$
STEP 2: Do for $t = 0, 1, \ldots, T-1$:
STEP 2a: Solve equation (71) to obtain $x_t^n = X_t^\pi(R_t^n, \hat{Q}_{t+1}^{n-1})$ and the duals $\hat{q}_t^n$ of the resource constraint (72).
STEP 2b: Update the resource state: R_{t+1}^n.
STEP 2c: Update the value function approximations using $\hat{Q}_t^n$.
STEP 3: Return the policy X_t^π and $\hat{Q}^N$.

Fig. 10. Single pass version of the adaptive dynamic programming algorithm

STEP 0: Initialize $\hat{Q}_t^0, \quad t \in \mathcal{T}$.
Set $n = 0$.
STEP 1: Do while $n \leq N$:
Choose $\omega^n \in \Omega$
STEP 2: Do for $t = 0, 1, \ldots, T-1$:
STEP 2a: Solve equation (71) to obtain $x_t^n = X_t^\pi(R_t^n, \hat{Q}_{t+1}^{n-1})$ and the duals $\tilde{q}_t^n$ of the resource constraint (72).
STEP 2b: Update the resource state: R_{t+1}^n.
STEP 3: Do for $t = T-1, T-2, \ldots, 1, 0$:
STEP 3a: Compute marginal value of a resource, $\bar{q}_t^n$, using $\hat{Q}_{t+1}^n$ and the optimal basis from the forward pass.
STEP 3b: Update the value function approximations, $\hat{Q}_t^n \leftarrow U^Q(\hat{Q}_t^{n-1}, \bar{q}_t^n, R_t^n)$.
STEP4: Return policy X_t^π and $\hat{Q}^N$.

Fig. 11. Double pass version of the adaptive dynamic programming algorithm

The single pass version of the algorithm is the easiest to implement, but it may not work the best. The problem is that it takes a full forward iteration to pass information back one time period. An alternative is to use a two-pass version of the algorithm, where there is a forward pass making decisions, and a backward pass updating dual variables. This version is described in figure 11.

These algorithms do not describe specifically how to update the functional approximations $\hat{Q}^n$. For the single-pass version of the algorithm, this updating process is identical to that used for the two-stage problem. We simply use the dual variables for the flow conservation

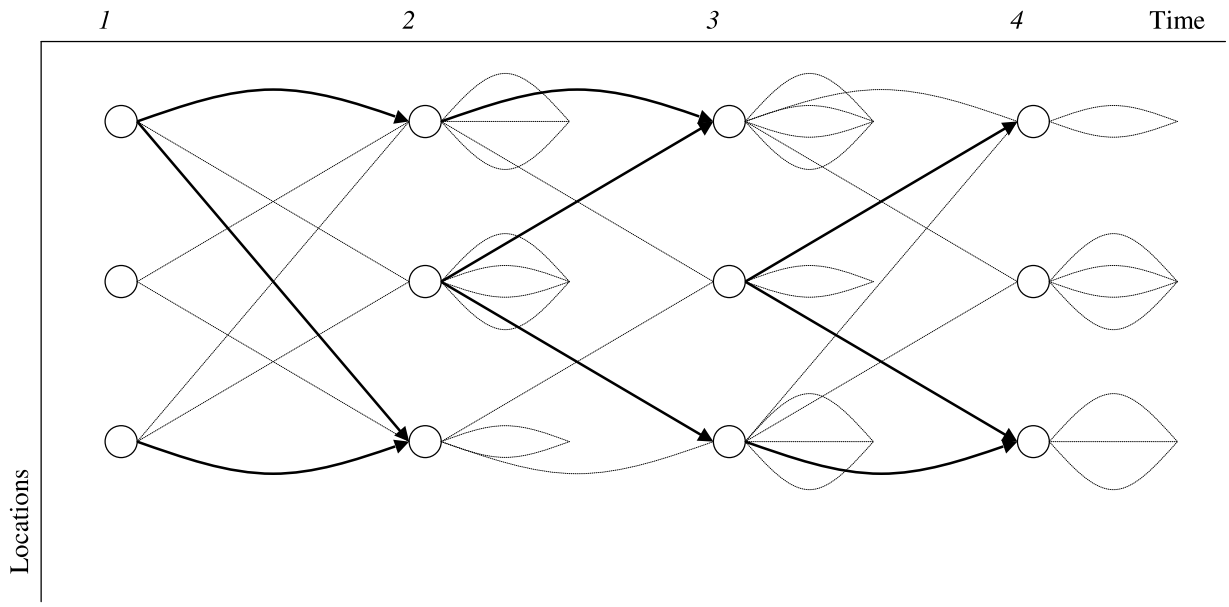

Fig. 12. Optimal network basis from the forward pass

constraint to update $\hat{Q}^n$ just as we did in section 5.5.2. As we have seen, there are a number of ways to update the value function, so we represent this in general using the notation:

$$\hat{Q}_t^n \leftarrow U^Q(\hat{Q}_t^{n-1}, \bar{q}_t^n, R_t^n)$$

Updating the value function in the two-pass version is a bit more involved, but the payoff is an updating process that possesses one particularly nice theoretical property. In addition, it appears to work better in practical applications. For the pedagogical purposes of this chapter, we are going to outline the basic idea graphically. Recall that we are solving sequences of pure networks at each time t. At each point in time, we obtain not only an optimal solution but also an optimal basis. Figure 12 shows the sequence of optimal bases over three time periods. Recall that the network structure of our one-period problems consists of links representing decisions in time period t plus links that represent our piecewise linear value functions. We are interested only in the portion of the basis that consists of links in time period t (with coefficients from the vector c_t), and not in the links which represent the approximation $\hat{Q}$. We note that as a result of our network structure, each basis path from a resource node consists of one or more links in time period t, finally ending in a node in a future time period.

After building the basis paths in the forward simulation, we now have a set of paths extending through the entire horizon. We then compute the cost of a path from a resource

node t for attribute vector a until the end of the horizon. Let $\bar{q}^n_{ta}$ be the cost of the path from resource node a at time t until the end of the horizon along this basis path. These path costs have a very nice property. Let:

$$F^\pi_t(R_t, \omega^n) = \sum_{t'=t}^{T} c_t X^\pi_t(I_t(\omega^n)) \tag{75}$$

be the costs of a policy π (determined by the functional approximation $\hat{Q}^n$) for outcome ω^n in iteration n, starting in time period t. Then we have:

Theorem 2 *Let $\bar{q}^n_t = (\bar{q}^n_{ta})_{a\in\mathcal{A}}$ be the vector of path costs from time t to the end of the horizon, given outcome ω^n and functional approximations $\{\hat{Q}^n_t\}_{t\in\mathcal{T}}$ computed from a backward pass. Then $\bar{q}_t$ satisfies:*

$$F^\pi_t(R_t, \omega^n) - F^\pi_t(R'_t, \omega^n) \leq \bar{q}^n_t \cdot (R_t - R'_t)$$

Furthermore, if the basis paths in each period t are flow augmenting paths into the supersink, then $\bar{q}^n_t$ is a right gradient of $F^\pi_t(R_t, \omega^n)$.

This is a very nice result. These paths are not too hard to compute and provide accurate estimates of the future value of a resource. It turns out that the ability to compute right derivatives is very important. If we just use the dual variables, we overestimate the value of a resource in the future, producing unwanted empty moves.

With our ability to compute valid stochastic gradients of the cost function, we are ready to apply all the tricks we learned for two-stage problems for computing the approximate value functions, $\hat{Q}^n_t$. The forward pass/backward pass logic is easy to execute and computationally tractable. The Monte Carlo sampling logic avoids problems with expectations. This almost looks too good to be true.

Multistage problems, however, have a few more surprises for us. We begin by discussing the problem of computing expectations, even when we use approximations of the recourse function. We next focus on the issue of problem structure. Keep in mind that transportation problems can be very large, and we are still interested in integer solutions. The first challenge that arises in multistage problems is that resources are reusable (box cars do not simply vanish after we use them). This introduces structural problems that did not occur with two-stage problems, which we review in the context of both single and multicommodity flow problems. We then briefly discuss one of the more annoying, but unavoidable, features of transportation problems: multiperiod travel times.

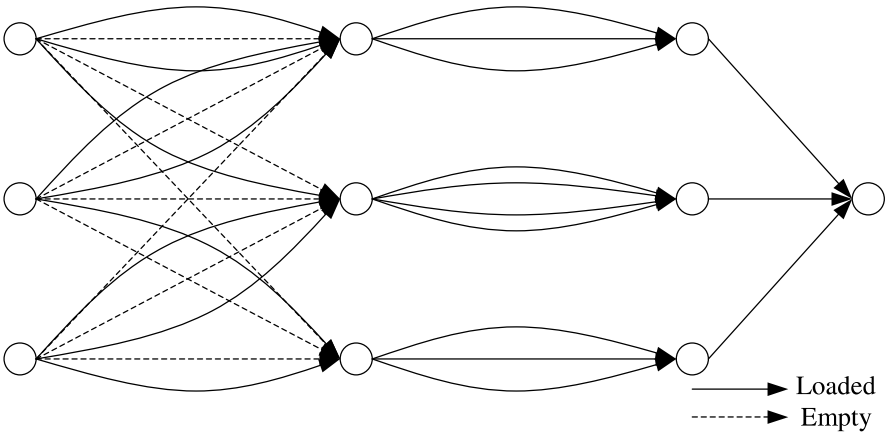

Fig. 13. Network structure of a one-period single commodity problem

6.3 Single commodity problems

The difference between the two-stage problem and the one-period problem in a multistage application is that the assignment of a resource to a task produces a resource in the future. In two-stage problems, once the car was assigned to an order in the second stage, it vanished from the system. In multistage problems, we find that we have to solve the type of network depicted in figure 13, which is a pure network. This can be solved with commercial LP solvers or specialized network algorithms. Assuming that the functions $\hat{Q}^n_{t+1}(R)$ is piecewise linear, with breakpoints defined for integer values of R, our pure network has integer data (upper bounds and resources) and as a result simplex-based algorithms will return integer solutions.

Pure networks have another attractive property. Linear programming codes will return a dual variable $\tilde{q}_{at}$ for the resource constraint (72). It is far more desirable to obtain explicit right and, if possible, left gradients, which we can denote $\tilde{q}^+_t$ and $\tilde{q}^-_t$. With pure networks, left and right gradients can be found by solving flow augmenting path problems into and out of (respectively) the supersink. A right gradient gives us a precise estimate of a particular slope of the recourse function. The computation of explicit left and right gradients is very important in problems with large attribute spaces, where the values of R_{ta} can be relatively small. If $\mathcal{A}$ is small relative to the number of resources being managed (implying that the values of R_{ta} are much greater than one), then the issue is unlikely to be noticeable.

The pure network structure actually arises in a larger class of problems. Assume that we are modeling resources with attribute a, and recall from section 3 that $a^M(t, a, d)$ is the *terminal attribute function*, giving the attributes of a resource with attribute a after

decision d has been applied to it. Now define:

Definition 3 *A resource allocation problem has the Markov property if* $a^M(t, a, d) = a^M(t, a', d),\ \forall a, a' \in \mathcal{A}$.

The Markov property for resource allocation problems implies that the attributes of a resource after it has been acted on is purely a function of the decision. Classical single commodity flow problems exhibit this property because a truck in location i which is then assigned to move a load from i to j, is now a truck at location j. If all the trucks are the same, then the attribute of the truck (its location) is purely a function of the attribute of the decision which was to move a load to j. If there were different types of trucks that could serve the load (a multicommodity problem), then the attribute of the truck after moving the load continues to have the attribute of the truck before the load (a characteristic that has nothing to do with the decision).

It is apparent that classical multicommodity problems which allow substitution of different commodity types for the same task will never have the Markov property, but multiattribute problems (where the attribute vector a consists purely of dynamic attributes), can possess this property. Consider a chemical trailer that can move basic argon gas or purified argon gas. Only "clean" trailers can move purified argon gas. A clean trailer can move basic gas, but then it is no longer clean. There is no substitution for purified argon gas, and any truck moving basic argon gas is no longer clean (although it can be put through a cleansing process). Thus, the attributes of the truck after a trip are determined completely by the characteristics of the trip.

6.4 Multicommodity problems

When we encountered single commodity problems, we found that the single period problems were pure networks if the approximation of the recourse function were linear, or piecewise linear, separable. Now we consider what happens when we try to solve multicommodity problems. Recall that we let $\mathcal{I}$ be a set of locations and $\mathcal{K}$ be the set of commodities. We follow the standard notation of multicommodity flow problems and let R_{ti}^k be the number of resources of type k in location i, and let x_{tdi}^k be the number of resources of type k in location i that we act on with decision d at time t.

Section 6.4.1 shows that multistage, multicommodity problems are especially easy to solve if we use linear approximations for the recourse function. The only issue is that the solution quality is not that high. Then, section 6.4.2 outlines the complications that arise when we use piecewise linear, separable approximations.

6.4.1 The case of linear approximations

When we replace the value function $Q_t(R_t)$ with an approximation $\hat{Q}_t(R_t)$, we obtain the decision function:

$$X_t^\pi(I_t) = \arg\max_x \sum_{i\in\mathcal{I}} \sum_{d\in\mathcal{D}} c_{tid}x_{tid} + E\left\{\hat{Q}_t(R_t(x_t))|R_{t-1}\right\} \tag{76}$$

If we use a linear approximation for $\hat{Q}$, then equation (76) reduces to:

$$X_t^\pi(I_t) = \arg\max_x \sum_{i\in\mathcal{I}} \sum_{d\in\mathcal{D}} c_{tid}x_{tid} + \sum_{t'>t} \sum_{j\in\mathcal{I}} \hat{q}_{t,jt'}\left(R_{t,jt'}(x_t,\omega_t)\right) \tag{77}$$

$$= \arg\max_x \sum_{i\in\mathcal{I}} \sum_{d\in\mathcal{D}} c_{tid}x_{tid} + \sum_{t'>t} \sum_{j\in\mathcal{I}} \hat{q}_{t,jt'}R_{t,jt'}(x_t,\omega_t) \tag{78}$$

where:

$$R_{t,jt'} = \sum_{i\in\mathcal{I}} \sum_{d\in\mathcal{D}_i} \delta_{t',j}(t,i,d)x_{tid} \tag{79}$$

Substituting equation (79) into (78) gives:

$$X^\pi(I_t) = \max_x \sum_{i\in\mathcal{I}} \sum_{d\in\mathcal{D}} c_{tid}x_{tid} + \sum_{t'>t} \sum_{j\in\mathcal{I}} \hat{q}_{t,jt'} \sum_{i\in\mathcal{I}} \sum_{d\in\mathcal{D}} \delta_{t',j}(t,i,d)x_{tid} \tag{80}$$

$$= \max_x \sum_{i\in\mathcal{I}} \sum_{d\in\mathcal{D}} c_{tid}x_{tid} + \sum_{i\in\mathcal{I}} \sum_{d\in\mathcal{D}} \left(\sum_{t'>t} \sum_{j\in\mathcal{I}} \delta_{t',j}(t,i,d)\hat{q}_{t,jt'}x_{tid}\right) \tag{81}$$

It is easy to see that:

$$\sum_{t'>t} \sum_{j\in\mathcal{I}} \delta_{t',j}(t,i,d)\hat{q}_{t,jt'}x_{tid} = \hat{q}_{t,i^M_{tid},t+\tau^M_{tid}}x_{tid} \tag{82}$$

where i^M_{tid} is our terminal attribute function (using location indices instead of attribute vectors) and τ^M_{tid} is the time required to complete the decision. This allows us to reduce (81) to:

$$X^\pi(I_t) = \max_x \sum_{i\in\mathcal{I}} \sum_{d\in\mathcal{D}} \left(c_{tid} + \hat{q}_{t+1,i^M_{tid},t+\tau^M_{tid}}\right)x_{tid} \tag{83}$$

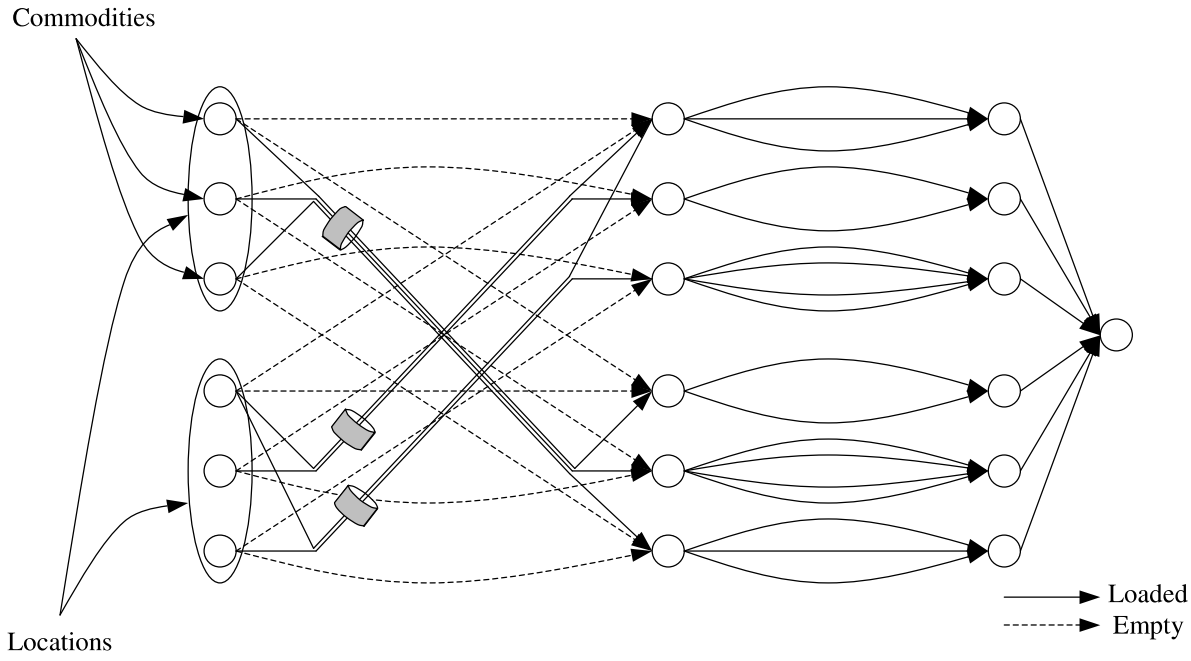

Fig. 14. Network structure of a one-period multicommodity problem

Equation (83) demonstrates that a linear approximation of the recourse function is the same as adding a price to the cost of each assignment, with the same structure as the original one-period problem. So, solving multicommodity flow problems with linear approximations is no harder than solving single commodity problems.

6.4.2 The case of nonlinear approximations

The situation is somewhat different when we are using nonlinear functional approximations. Our one-period problem now takes on the network structure shown in figure 14 which is itself a multicommodity flow problem, if we are using nonlinear functional approximations. We note, however, that these are not especially large multicommodity flow problems, since they are for a single time period. Perhaps the most significant practical problem that might arise is the loss of integrality. In fact, we have found that a commercial LP package solving a continuous relaxation of the problem returns integer solutions 99.9 percent of the time (the rare instance of a fractional solution is quickly solved with branch and bound or simple rounding heuristics).

Perhaps the more practical challenge of multicommodity flow problems is that we do lose the ability to find left and right gradients using flow augmenting path calculations. As we pointed out before, this is not necessary for all problem classes, but we have worked on problems with large attribute spaces where dual variables from the LP solver are simply

not good enough. In this case, we are resorting to performing numerical derivatives (incrementing the right hand side and solving the problem again). Since the optimal basis is often optimal for the perturbed problem, this procedure can be quite fast.

6.5 The problem of travel times

One of the most annoying characteristics of transportation is the property that it takes time to complete a decision. Using the vocabulary of discrete time models, we refer to these problems as having "multiperiod" travel times. More generally, we would refer to these as "multiperiod transfer times," since there are many activities in transportation that do not actually involve moving from one location to another (drivers have to go on rest, trailers have to be cleaned, locomotives have to be maintained). But, the concept of traveling between locations is easy to visualize.

The implication of multiperiod travel times is that after acting on a resource at time t, the resource is committed to an activity in time period $t+1$ and we cannot act on it. At the same time, we cannot ignore it, because it will eventually complete its movement, and we have to take this into account when we make decisions in time period $t+1$. Figure 15 illustrates the issue in the context of fleet management. Assume that we will need containers at location **c** at time $t = 6$, and we can move them there from either **a**, which is five time periods away, or from **d**, which is only two time periods away. Because **a** is farther away, we will first look at the problem of the shortage of containers at **c** for time $t = 6$ at time $t = 1$. At this point, we would not have considered moving containers from location **d**, since this decision would not be made until $t = 4$. Seeing the shortage, we might move containers the longer distance from **a** to **c**, rather than waiting until time $t = 4$ and moving them from the closer location at **d**.

The modeling of multiperiod travel times is the same as the modeling of lagged information processes. Earlier in the chapter, we introduced the notation $R_{tt'}$ which gives the vector of resources that we know about at time t which become actionable at time t'. The difference between t' and t is the information lag. Lags can arise when customers call in orders in advance. In the case of our rail car distribution problem, it can arise when a shipper tells the carrier that a freight car will become empty in three days. Information lags also arise whenever the travel time from one location to another is more than one time period.

The problem of multiperiod travel times is unique to multistage stochastic models, and furthermore it is unique to the usage of nonlinear functional approximations. With a nonlinear function, location **a** "sees" the steeper part of the slope of the function at **c**, since we have not yet made the decision to move cars from **d**. The result is something that we call the "long haul bias," which arises in any application where resources can be

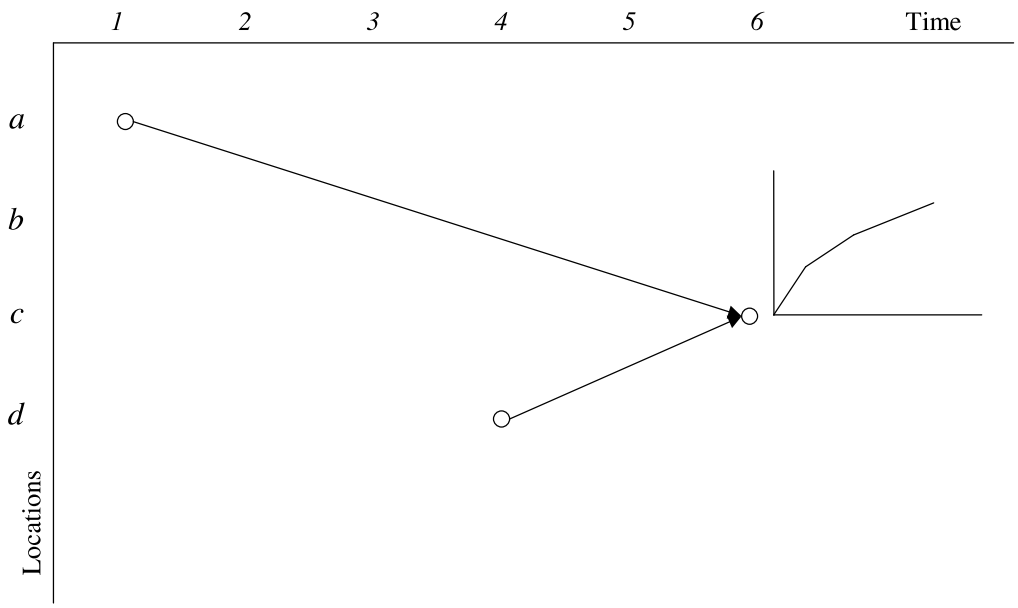

Fig. 15. Multiperiod travel times imply that decisions to move vehicles at different points in time can produce available capacity at the same time in the future.

committed in the future.

The standard solution for problems of this type is to use an augmented state variable. Assume that a container started moving from i to j, departing at time t, on a trip that requires time τ_{ij}. Now let the variable s be the remaining time in the trip, so at time $t+1$, we would have $s = \tau_{ij} - 1$. Using our multiattribute notation, the remaining trip time s simply becomes a new dimension of the attribute vector a. Given this representation, we can solve multiperiod problems using the same techniques that we have described up to now.

This particular solution creates practical problems in transportation applications. Problems in trucking and rail, for example, can have trip times that range from 30 minutes to three days (for movements from the midwest to the west coast). We can often work with time steps of two or four hours, producing trips that are often 10 to 20 time periods in length. The result is an attribute space $\mathcal{A}$ that is now approximately 10 times bigger (a dramatic increase in the size of the problem). Since a single movement is now broken into 10 or 20 time steps, pure forward pass algorithms become completely unworkable, although we can partly overcome the slow backward communication of duals using shortcuts that take advantage of the properties of the problem.

The augmented state representation has to be viewed as a brute force solution to the problem of multiperiod travel times which can actually hide nice structural properties.

For example, it is not readily apparent using this representation that the problem of multiperiod travel times vanishes when we use linear functional approximations. When $\hat{Q}_{C,t+6}$ is a linear function, both locations **a** and **d** "see" the same slope. If they both send containers in response to an attractive slope, then the sample gradient of the function at location **c** will be reduced, and the location will become less attractive. Over sufficient iterations, the model should discover the slope that attracts capacity from closer locations but not farther ones.

It is beyond the scope of our presentation to fully describe the solution to the "multiperiod travel time" problem when using nonlinear functional approximations, but we note that it involves replacing the single functional approximation $\hat{Q}_{t'}$ with a family of functions $\hat{Q}_{tt'}$ which are used to describe the impact of decisions made at time t on future points in time $t' > t$. This approach produces solutions that are comparable in quality to those obtained using nonlinear approximations with single-period travel times, but as of this writing, the theory behind this approach is immature.

We again see that linear approximations avoid complexities that arise in the context of nonlinear approximations. But, the jury is still out regarding the approach that will produce the best results in the laboratory, and implementable results in the field.

7 Some experimental results

There is surprisingly little work comparing different stochastic programming approaches. This is especially true of multistage problems, but it is even true of the much more mature two-stage problem. Furthermore, the work on two-stage problems has not been done explicitly in the context of resource allocation problems (they are simply characterized as two-stage linear programs) which makes it difficult to generate a library of datasets which focus on the specific dimensions of resource allocation problems (such as number of locations, number of commodity types, repositioning costs in the second stage, and so on). As a result, we do not have a standard library of test datasets for either two-stage or multistage resource allocation problems.

This chapter has described the use of a relatively new class of approximation strategies that are especially well suited to resource allocation problems. These approximations focus on using linear or separable, nonlinear approximations of the recourse function. Many resource allocation problems require integer solutions. Separable, nonlinear functions can be constructed as piecewise linear approximations which produces first stage problems that are either pure networks, or integer multicommodity flow problems with very tight LP relaxations. In this section, we provide some preliminary comparisons between these

approximation strategies and a variant of Benders decomposition called the CUPPS algorithm.

Section 7.1 describes some experiments that were performed in the context of two-stage problems. Section 7.2 then presents some results for multistage problems.

7.1 Experimental results for two-stage problems

Virtually all problems in transportation and logistics involve multistage problems, but as our discussion has demonstrated, multistage problems are typically solved as sequences of two-stage problems. As a result, we have to begin with an understanding of how well we can solve two-stage problems.

We undertook a series of preliminary experiments focusing on questions regarding rate of convergence, scalability and solution quality. We used a randomly generated dataset (which allowed us control over its characteristics), and compared SPAR (which uses separable, piecewise linear approximations) and CUPPS (based on Benders decomposition). Our evaluation strategy consisted of running each algorithm for a fixed number of training iterations, and then comparing solution quality over a series of testing iterations.

Our datasets were created very simply. N locations were uniformly generated over a 100 $\times$ 100 square. Initial supplies of resources were randomly generated and spread around these locations. The resources and demands were generated in such a way that ensured that the expected number of resources equaled the expected number of demands (our work has shown that this is when the problems are the most difficult, the most interesting and the most realistic). Demands (in the form of loads of freight to be moved) were randomly generated with both an origin and a destination, where the contribution from covering a demand was set at \$1.5 per mile, where the length of the load is given by the distance from the origin to the destination. Transportation costs in the first stage are fixed at \$1 per mile, while transportation costs in the second stage are fixed at \$2 per mile. This provides an incentive to reposition in the first stage before we know what the demands are.

Our first experiments studied the rate of convergence of each algorithm on a dataset with 30 locations. Figure 16 shows the objective function for SPAR and CUPPS averaged over 50 testing iterations, as a function of the number of training iterations. With 950 training iterations, the methods are virtually identical. However, as the number of training iterations diminishes, SPAR seems to perform better, suggesting that it has a faster rate of convergence.

This conclusion is supported in figure 17 which compares SPAR and CUPPS as a function

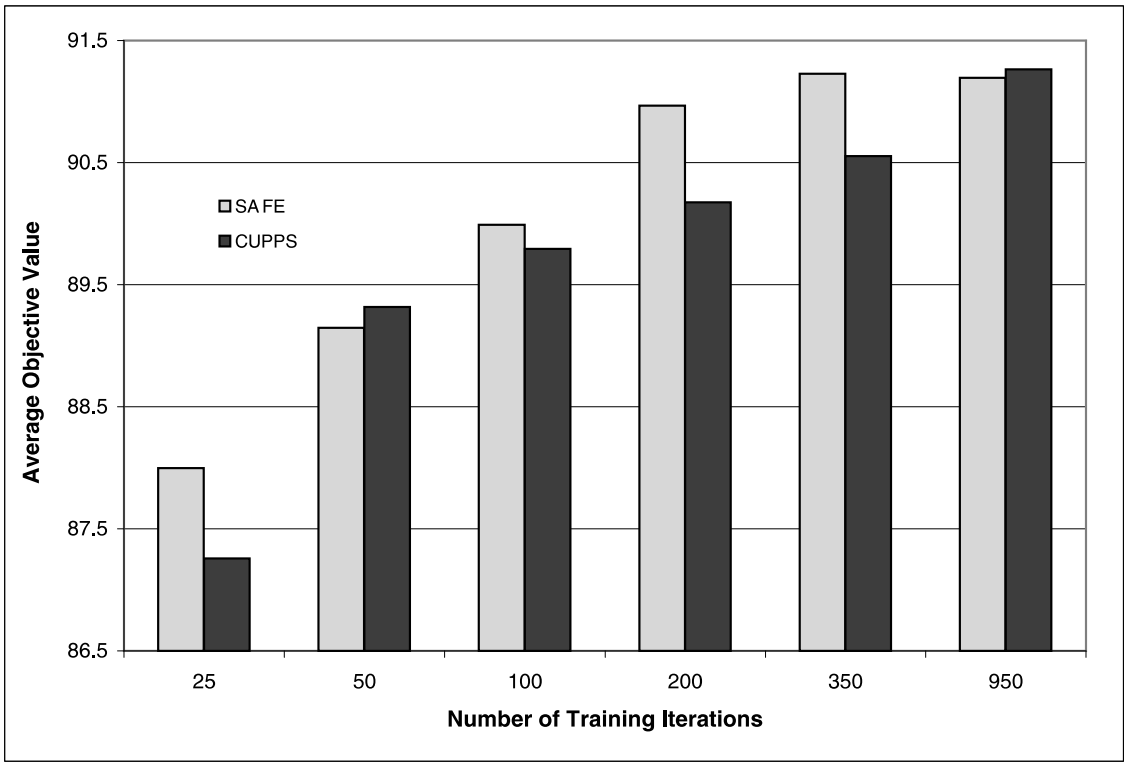

Fig. 16. The effect of the number of training iterations on SPAR and CUPPS, illustrating the faster convergence for SPAR

of the number of locations. For each run, we used 200 training iterations, and the algorithms were run on problems with 20, 30, 40 and 90 locations. The results show that SPAR and CUPPS work similarly for smaller problems, but that SPAR works better (with 200 training iterations) for larger problems. This suggests that the SPAR-class algorithms exhibit a faster rate of convergence, especially as the problem size grows.

We finally looked at the results for every observation within the test sample to get a sense of the distribution of the difference between SPAR and CUPPS. To our surprise, we found that SPAR and CUPPS provide almost identical results for every outcome for smaller datasets. For larger datasets, SPAR outperformed CUPPS on *every* outcome.

7.2 Experimental results for multistage problems

The solution of multistage problems for our applications consist of solving sequences of two-stage problems. The question now is, how well does this work? Should we expect that an optimal or near-optimal algorithm for two-stage problems will work similarly on

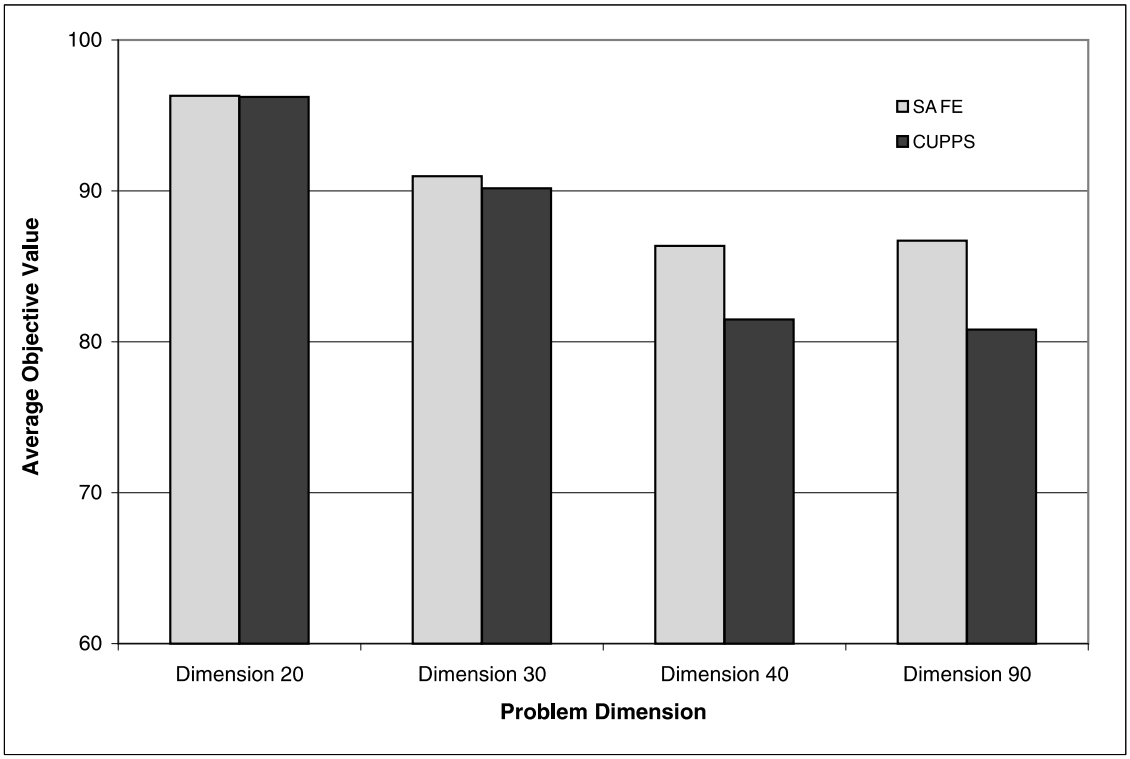

Fig. 17. SPAR vs. CUPPS for two-stage problems, illustrating better results for SPAR when the problem size grows

multistage problems?

The biggest difference between two-stage and multistage problems for our problem class is that the sequences of two-stage problems that we solve in a multistage setting have random initial starting states. When we solve a two-stage problem, the initial state is (normally) deterministic. This means that the optimal solution to the first stage is deterministic, which means that our approximation for the second stage has to be accurate only in the vicinity of the optimal solution of the first stage. In the case of multistage problems, the initial resource state at some time t in the future depends on previous decisions, while the approximation of the recourse function for this problem is fixed, and must perform well over a range of initial states. As a result, the demands on the accuracy of the recourse function for the second stage are much higher.

A major difficulty that arises in the evaluation of approximations for multistage problems is identifying a good benchmark. Optimal solutions are simply not obtainable, and tight bounds are not readily available. For this reason, we use two strategies. First, it is useful to see how well a stochastic algorithm works on a deterministic problem. This need arises since it is typically the case that a company will want to test how well the algorithm works

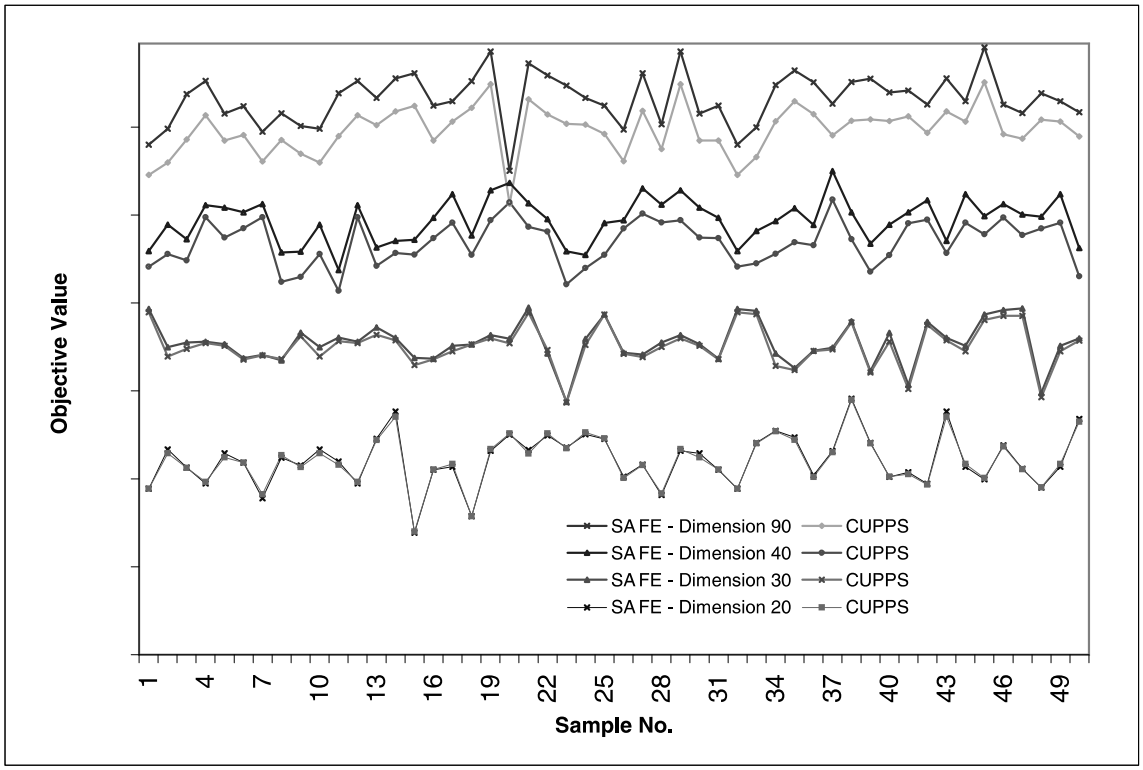

Fig. 18. SPAR outperforms CUPPS as the problem size grows for every outcome in the testing dataset

	Simulation Horizon		
Locations	**15**	**30**	**60**
20	100.00%	100.00%	100.00%
40	100.00%	99.99%	100.00%
80	99.99%	100.00%	99.99%

Table 1
Percentage of integer optimal value obtained using SAFE for second set of deterministic experiments with single-period time windows (network problems)

by running it on past history (which is deterministic). Deterministic formulations tend to be the benchmark, and if a stochastic algorithm does not work well on a deterministic problem, it raises the question of how it can be a good method for a stochastic problem.

The second strategy we use is to compare against deterministic rolling horizon procedures using stochastic data. Again, this is the most common strategy used in engineering practice for solving stochastic problems.

We first ran the SPAR algorithm on a deterministic, single commodity problem which can

be formulated as a pure network. A significant assumption is that the time at which a load had to be moved was fixed (so-called tight time windows). Table 1 reports these results for problems with 20, 40 and 80 locations. These results indicate that the algorithm is effectively returning optimal solutions.

We then ran the algorithm on four stochastic datasets and compared against a rolling horizon procedure (RHP). The RHP used a point forecast of the future to make decisions for the current time period. Tests were run with different planning horizons to ensure that we were using the best possible planning horizon for the RHP. The results are shown in figure 19 which shows that the SPAR algorithm is producing results that are significantly better than a deterministic RHP.

Further research is needed before we understand the true value of a stochastic formulation. Our rolling horizon experiments were performed assuming that there was no advance information. The value of a stochastic model also depends on the economics of making the wrong decision (the recourse).

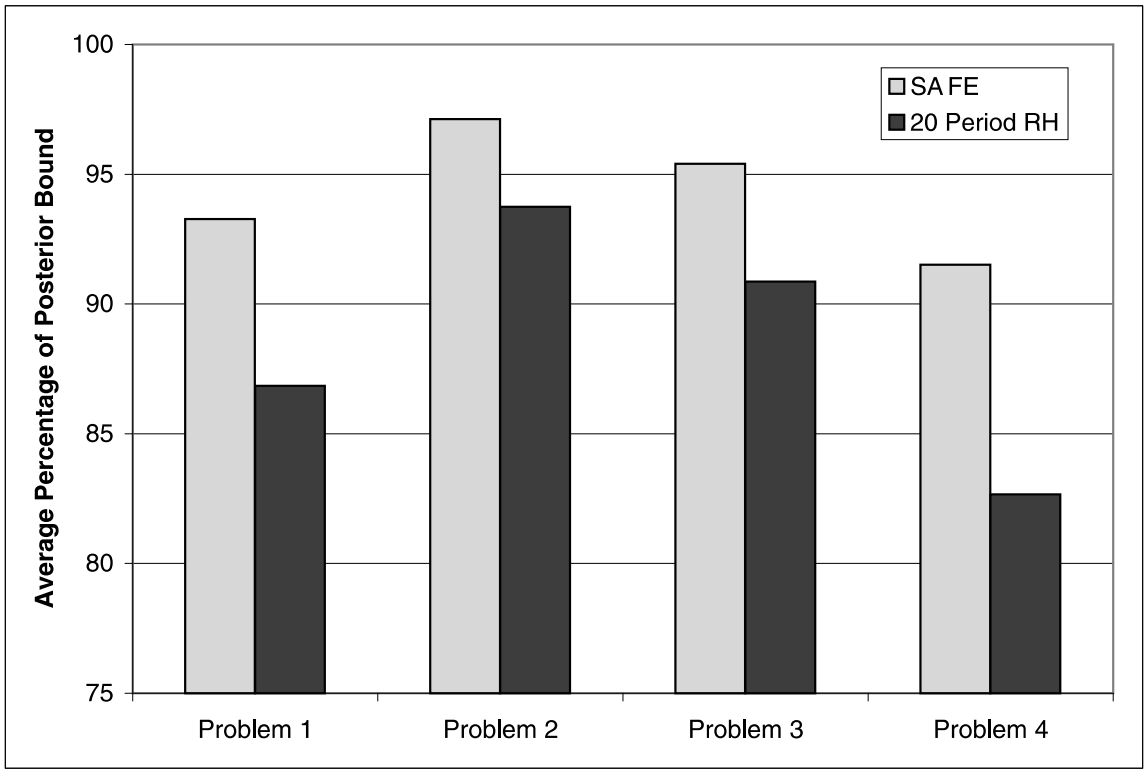

Fig. 19. Comparison of SPAR approximations to rolling horizon procedures for 20 location datasets and different substitution rules

8 A list of extensions

This chapter has introduced a basic problem class, discussing along the way practical algorithmic strategies, but steadily introducing issues that address the richness of transportation problems. Our notational framework, which at first may seem clumsy to researchers accustomed to even simpler notation, is designed to provide a natural bridge between the classical notation of mathematical and stochastic programming, but provide for some of the issues that arise in practice. Our progression of problems, from two-stage to multistage, from single commodity with no substitution through multicommodity and heterogeneous resource allocation problems, was designed to bring the reader through a list of issues that add to the realism of the model being considered.

We have brought the reader to the edge of realistic problems that rise in practice, but real problems of the type that arise in freight transportation have even more surprises to delight and frustrate the research community. In this section, we provide a hint of problems that remain to be addressed. All of the issues listed below represent forms of inaccurate information.

a) Random travel times - Not only are there multiperiod travel times, it is normally the case that the travel time is random. A travel time may be uncertain even when a decision is made.
b) Advance information - Customers place orders in advance. Railroads might let you know that they will give you empty cars in three days. A driver might tell you that he will start in a week. We need to model the presence of information that is known, but not actionable.
c) Demand backlogging and the "now vs. later" problem - If we do not serve the customer request now, we may be able to serve it later (at a price). We can assign a driver to a distant load now, or wait for a closer load to be called in later. We need to identify when we should make a decision now (to serve a request) or wait until later.
d) Multiple, reusable layers - Perhaps the most challenging problem is the presence of multiple resource layers. (Recall that customer demands can also represent a kind of resource layer). The simplest example arises with backlogging demands: if we do not serve a demand now, it is still in the system in the next time period. In freight transportation, we may have to manage drivers, tractors and trailers, or locomotives, boxcars and crews. It is possible to model some freight transportation operations with four or five layers, and most of them are reusable.
e) User noncompliance - An overlooked dimension of most models is that what the model is recommending is not what is being implemented. So the costs and contributions that we are adding up in the computer are not the costs and contributions we are getting in the field. The difference between what a model recommends and what is implemented

in the field is a source of randomness that draws into question the value of so-called optimal solutions that ignore this source of randomness.

f) Multiagent control - Large problems are typically broken into smaller problems which are managed by different people. Decisions made by one person need to anticipate the impact on another. But it is impossible to predict what someone else will do with certainty. This is not the same as the user compliance problem, but it is another instance of solving a problem where the randomness is in predicting what someone will do.

g) Data errors - We all know about data problems and recognize that we have to fix them, but we overlook the fact that the presence of data errors is again a source of noise. If data errors are going to require humans to override model recommendations, then so-called "optimal" solutions are no longer optimal (even within a single stage).

h) Incomplete information - Random variables arise when we have information that is not known now, but can be measured later. There are problems where information is unknown but can never be measured directly, which means we could never estimate a probability distribution. But the missing information is captured in historical databases of past decisions.

9 Implementing stochastic programming models in the real world

We close our discussion of stochastic problems by raising some of the issues that arise when we try to implement stochastic programming models in practice. In the beginning of this chapter, we made the argument that explicit models of uncertainty produce more realistic behaviors, often exactly the behaviors that humans will mimic (perhaps with less precision) but deterministic models will overlook.

Humans have an innate ability to deal with imperfect information and noise. We allow more time to make an appointment. We provide slack time in airline schedules to allow for possible delays. We own extra trucks and locomotives to account for breakdowns, congestion and spikes in demand. Trucking companies have large rooms of people planning operations who spend 90 percent of their time collecting and verifying data, and only 10 percent actually making decisions. We would expect that planners would quickly embrace models which capture the uncertainty that they spend so much time dealing with.

Surprisingly, this is not the case. Stochastic models arise because of the need to forecast an activity in the future, a forecast that is made with uncertainty. And yet in the business world, the word "forecast" is synonymous with the concept of a "point forecast." When we ask for a forecast of the number of box cars needed in a region next week, we do not want to be told "somewhere between 70 and 100." When we ask for a forecast, we expect an answer such as "85."

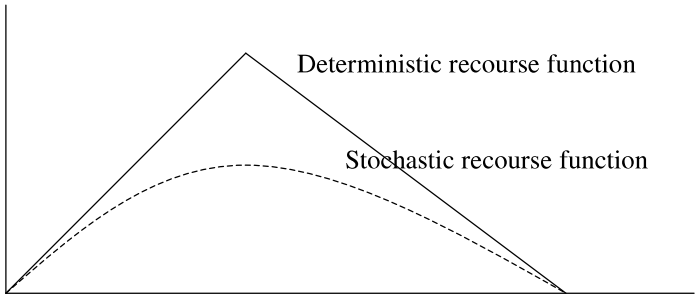

Fig. 20. Recourse functions for deterministic and stochastic functions

At the heart of any stochastic programming application is a *distributional forecast*, an explicit recognition that a range of outcomes is possible. As much as people in operations have learned to deal with uncertainty, they uniformly have difficulty working with models which capture this uncertainty. Often this is because they are looking either for a specific recommendation of what to do right now (which even a stochastic model should do) or a *plan* of what should be done in the future. It can be said that a plan is a (point) forecast of a decision. Even for events in the future, people want to be told what is going to happen (even if they realize a plan has uncertainty, they are still looking for a specific estimate of what is going to happen).

Consider the practical difficulty of testing a stochastic model. In a deterministic model, we expect the model to move exactly the number of freight cars that are needed. If a deterministic model moved 60 cars to satisfy orders for 50, we might reasonably conclude that there was some sort of bug in the software. Yet this might be exactly the right answer for a stochastic model. But how do we know that we are getting optimal behavior, or simply the result of a programming error?

For this reason, we have found that we usually first test a stochastic programming model by solving a deterministic problem, and comparing the solution against a standard solver for deterministic problems. It seems obvious that an algorithm that can solve a stochastic problem should also be able to solve a deterministic problem, but this can be harder than it looks. Figure 20 shows a recourse function for deterministic and stochastic models. As a rule, stochastic problems are smoother and better behaved. As a result, linear approximations can work quite well. However, these same techniques will not work as well on the sharply angled function produced when we replace our range of possible outcomes with a point forecast. While we can argue that we should not have to test our stochastic algorithm on a deterministic model, this is a powerful debugging and testing tool, and we have found that it is necessary to develop techniques that work well on deterministic as well as stochastic problems.

10 Bibliographic notes

The techniques in this chapter have their roots in stochastic approximation methods (Robbins & Monro (1951), Blum (1954), Dvoretzky (1956), Gladyshev (1965)), stochastic gradient methods (Ermoliev (1988)), general stochastic linear programming (Birge & Louveaux (1997), Infanger (1994), Kall & Wallace (1994)) and dynamic programming (both classical methods, reviewed in Puterman (1994), and approximate methods, such as those covered in Bertsekas & Tsitsiklis (1996), Sutton & Barto (1998)). For reviews of these topics, the reader is referred to the introductory chapters in this volume.

Applications in transportation and logistics represented some of the earliest motivations for stochastic programming. Dantzig (1955) used fleet management (in an airline setting) as an early motivation for stochastic programming. Ermoliev et al. (1976) formulated the planning of empty shipping containers as a stochastic program.

There has been a rich history of research in fleet management in the context of the "car distribution problem" of railroads. Most of this work consists of deterministic linear programming models (Feeney (1957), Leddon & Wrathall (1967), Gorenstein et al. (1971), Misra (1972), White (1972), Herren (1973), Herren (1977), White & Bomberault (1969), Haghani (1989), Mendiratta (1981), Mendiratta & Turnquist (1982)). Dejax & Crainic (1987) provide a thorough review of the research in fleet management at the time, covering both rail and intermodal container applications. Crainic et al. (1993) provide a general stochastic, dynamic model for container distribution.

Jordan & Turnquist (1983) provide a stochastic formulation of the empty car distribution problem. In their model, a car could be assigned to at most one demand, and cars could not be repositioned more than once. This structure allowed the problem to be formulated as a nonlinear programming problem. Powell (1986) extended this methodological approach, using the trucking industry as the context, to multistage problems with reusable resources. This formulation involved forming deterministic decision variables which specified the percentage of the supply of trucks at a node that would be moved loaded or empty from one location to another. This line of research, however, was unable to properly model the recourse strategy where a truck might be assigned to choose from a set of loads going out of a location to various destinations.

Powell (1987) solved this problem in a way that produced a pure network formulation for the first stage subproblem (see Powell (1988) for an overview of different ways of formulating the problem). A major strength of the technique was that it produced nonlinear functions of the value of vehicles in the future. The ideas behind this research produced a series of articles that approximated the recourse function by using the structure of the underlying recourse function (Powell & Frantzeskakis (1992), Frantzeskakis & Powell

(1990), Powell & Cheung (1994*b*), Powell & Cheung (1994*a*), Cheung & Powell (1996)). These papers introduced concepts such as nodal recourse (the recourse is to allocate flow over different links out of a node), tree recourse (the recourse is to optimize flows over a tree) and other restricted recourse strategies aimed at approximating more complex problems. Although the results were promising, this line of research required approximating the future in a way that prevented the techniques from being applied to the most general (and realistic) problems.

Powell & A. (1998) (see also Carvalho & Powell (2000)) took a different tact and solved the problem as a sequence of network problems, using linear approximations of the value of the future. This approach was computationally quite easy, and scaled to much harder problems. Powell et al. (2002*b*) showed how the technique could be applied to the heterogeneous resource allocation problem, which is a more general problem than the multicommodity flow problem (the resource attribute space is much larger). The use of linear approximations to represent the future, however, produced instabilities that were solved by the use of control variables that limited the amount of flow that could be moved from one location to another. This worked very well for single commodity problems, but did not scale well to multicommodity or heterogeneous resource allocation problems.

Godfrey & Powell (2001) introduce an adaptive sampling technique, dubbed the CAVE algorithm, that produces nonlinear approximations of a recourse function using stochastic gradients. The method is easy to apply and produces piecewise linear, separable approximations of a recourse function. Furthermore, all it requires is the dual variable from a second stage problem, and does not require that the problem have any particular structure. Experimental work suggested that the algorithm might be optimal for two-stage problems, but at a minimum it produced results that were extremely close to optimal for both deterministic and specially structured stochastic problems. Godfrey & Powell (2002*a*) apply it to stochastic, multistage problems and demonstrate very good results relative to rolling horizon procedures. Godfrey & Powell (2002*b*) investigated the case of resource allocation where the travel time from one location to another can be multiple periods. A naive application of the CAVE algorithm produced extremely poor results, and a variation is suggested that provides results that are comparable to the single period travel time case.

Topaloglu & Powell (2002) applies similar techniques to stochastic multicommodity flow problems. This problem class introduces the additional complexity that multicommodity problems, combined with nonlinear approximations of the future, produce sequences of (usually integer) multicommodity flow problems. The method appears to work well on both both deterministic and multistage stochastic integer multicommodity flow problems.

The SPAR algorithm is part of a broader class of algorithms that use stochastic gradients but maintain structural properties such as concavity. This strategy was first suggested in

Godfrey & Powell (2001) as the CAVE algorithm, but the SPAR algorithm, introduced by Powell et al. (2002*a*), offers several provable convergence results and appears to also work better experimentally (see Topaloglu & Powell (2002) for the use of these techniques for multistage problems). Auxiliary function methods (Culioli & Cohen (1990) and Cheung & Powell (2000)) maintain concavity by starting with a concave function and using stochastic gradients to update the function (effectively tilting it).

References

Bertsekas, D. & Tsitsiklis, J. (1996), *Neuro-Dynamic Programming*, Athena Scientific, Belmont, MA.

Birge, J. & Louveaux, F. (1997), *Introduction to Stochastic Programming*, Springer-Verlag, New York.

Blum, J. (1954), 'Multidimensional stochastic approximation methods', *Annals of Mathematical Statistics* **25**, 737–744.

Carvalho, T. A. & Powell, W. B. (2000), 'A multiplier adjustment method for dynamic resource allocation problems', *Transportation Science* **34**, 150–164.

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

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

Crainic, T., Gendreau, M. & Dejax, P. (1993), 'Dynamic stochastic models for the allocation of empty containers', *Operations Research* **41**, 102–126.

Culioli, J.-C. & Cohen, G. (1990), 'Decomposition/coordination algorithms in stochastic optimization', *SIAM Journal of Control and Optimization* **28**, 1372–1403.

Dantzig, G. (1955), 'Linear programming under uncertainty', *Management Science* **1**, 197–206.

Dejax, P. & Crainic, T. (1987), 'A review of empty flows and fleet management models in freight transportation', *Transportation Science* **21**, 227–247.

Dvoretzky, A. (1956), On stochastic approximation, *in* J. Neyman, ed., 'Proc. 3^{rd} Berkeley Sym. on Math. Stat. and Prob.', Berkeley: University of California Press, pp. 39–55.

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

Ermoliev, Y., Krivets, T. & Petukhov, V. (1976), 'Planning of shipping empty seaborne containers', *Cybernetics* **12**, 664.

Feeney, G. (1957), Controlling the distribution of empty cars, *in* 'Proc. 10th National Meeting, Operations Research Society of America'.

Frantzeskakis, L. & Powell, W. B. (1990), 'A successive linear approximation procedure for stochastic dynamic vehicle allocation problems', *Transportation Science* **24**(1), 40–57.

Gladyshev, E. G. (1965), 'On stochastic approximation', *Theory of Prob. and its Appl.* **10**, 275–278.

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

Godfrey, G. & Powell, W. B. (2002*b*), 'An adaptive, dynamic programming algorithm for stochastic resource allocation problems II: Multi-period travel times', *Transportation Science* **36**(1), 40–54.
Godfrey, G. A. & Powell, W. B. (2001), '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.
Gorenstein, S., Poley, S. & White, W. (1971), On the scheduling of the railroad freight operations, Technical report 320-2999, ibm philadelphia scientific center, IBM.
Haghani, A. (1989), 'Formulation and solution of a combined train routing and makeup, and empty car distribution model', *Transportation Research* **23B**(6), 433–452.
Herren, H. (1973), 'The distribution of empty wagons by means of computer: An analytical model for the Swiss Federal Railways (SSB)', *Rail International* **4**(1), 1005–1010.
Herren, H. (1977), 'Computer controlled empty wagon distribution on the SSB', *Rail International* **8**(1), 25–32.
Infanger, G. (1994), *Planning under Uncertainty: Solving Large-scale Stochastic Linear Programs*, The Scientific Press Series, Boyd & Fraser, New York.
Jordan, W. & Turnquist, M. (1983), 'A stochastic dynamic network model for railroad car distribution', *Transportation Science* **17**, 123–145.
Kall, P. & Wallace, S. (1994), *Stochastic Programming*, John Wiley and Sons, New York.
Leddon, C. & Wrathall, E. (1967), Scheduling empty freight car fleets on the louisville and nashville railroad, *in* 'Second International Symposium on the Use of Cybernetics on the Railways, October', Montreal, Canada, pp. 1–6.
Mendiratta, V. (1981), A dynamic optimization model of the empty car distribution process, Ph.D. Dissertation, Department of Civil Engineering, Northwestern University.
Mendiratta, V. & Turnquist, M. (1982), 'A model for the management of empty freight cars', *Trans. Res. Rec.* **838**, 50–55.
Misra, S. (1972), 'Linear programming of empty wagon disposition', *Rail International* **3**, 151–158.
Powell, W. B. (1986), 'A stochastic model of the dynamic vehicle allocation problem', *Transportation Science* **20**, 117–129.
Powell, W. B. (1987), 'An operational planning model for the dynamic vehicle allocation problem with uncertain demands', *Transportation Research* **21B**, 217–232.
Powell, W. B. (1988), A comparative review of alternative algorithms for the dynamic vehicle allocation problem, *in* B. Golden & A. Assad, eds, 'Vehicle Routing: Methods and Studies', North Holland, Amsterdam, pp. 249–292.
Powell, W. B. & A., C. T. (1998), 'Dynamic control of logistics queueing network for large-scale fleet management', *Transportation Science* **32**(2), 90–109.
Powell, W. B. & Cheung, R. (1994*a*), 'A network recourse decomposition method for dynamic networks with random arc capacities', *Networks* **24**, 369–384.
Powell, W. B. & Cheung, R. (1994*b*), 'Stochastic programs over trees with random arc capacities', *Networks* **24**, 161–175.
Powell, W. B. & Frantzeskakis, L. (1992), 'Restricted recourse strategies for stochastic, dynamic networks', *Transportation Science* **28**(1), 3–23.
Powell, W. B., Ruszczynski, A. & Topaloglu, H. (2002*a*), Learning algorithms for separable approximations of stochastic optimization problems, Technical report, Princeton University, Department of Operations Research and Financial Engineering.
Powell, W. B., Shapiro, J. A. & Simão, H. P. (2002*b*), 'An adaptive dynamic programming

algorithm for the heterogeneous resource allocation problem', *Transportation Science* **36**(2), 231–249.

Puterman, M. L. (1994), *Markov Decision Processes*, John Wiley and Sons, Inc., New York.

Robbins, H. & Monro, S. (1951), 'A stochastic approximation method', *Annals of Math. Stat.* **22**, 400–407.

Sutton, R. & Barto, A. (1998), *Reinforcement Learning*, The MIT Press, Cambridge, Massachusetts.

Topaloglu, H. & Powell, W. B. (2002), Dynamic programming approximations for stochastic, time-staged integer multicommodity flow problems, Technical report, Princeton University, Department of Operations Research and Financial Engineering.

White, W. (1972), 'Dynamic transshipment networks: An algorithm and its application to the distribution of empty containers', *Networks* **2**(3), 211–236.

White, W. & Bomberault, A. (1969), 'A network algorithm for empty freight car allocation', *IBM Systems Journal* **8**(2), 147–171.