Fleet Management

Warren B. Powell and Huseyin Topaloglu

June, 2002

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

Abstract

Fleet management addresses the problem of managing fleets of trailers, containers, boxcars, taxicabs, locomotives and business jets. In many applications, requests to be moved from one location to another arrive randomly over time. Since it may take days to move from one location to the next, it is often useful (even necessary) to reposition equipment before a customer demand is known. We use the techniques of stochastic optimization to solve two-stage and multistage fleet management problems, using the problem of rail car distribution as the motivating application. Rail car distribution is characterized by a high degree of randomness: in the orders being placed by the customers, in the empty cars becoming available, and in the transit times between locations. Car distribution problems also introduce other issues that are not typically addressed in the classical stochastic programming literature, such as lagged information processes, the need for discrete solutions, and the challenges posed when the relevant attributes of a car are allowed to grow (a common progression when solving real problems). This chapter is intended to introduce the reader to a real problem, providing a bridge between the classical literature of stochastic programming and the modeling and algorithmic challenges that would arise while solving a real problem.

1 Introduction

The fleet management problem, in its simplest form, involves managing fleets of equipment to meet customer requests as they evolve over time. The equipment might be containers which hold freight (ocean containers, truck trailers, boxcars), equipment such as locomotives, truck tractors, taxicabs or business jets (companies in the fractional jet ownership business may own up to 1,000 jets). The equipment has to serve customers (people or freight) who typically want to move from one location to the next. We make the assumption throughout that a piece of equipment can serve one request at a time.

One of the challenges of fleet management is that customer requests arrive randomly over time, often requiring service within a narrow interval. Since it can take from several days to over a week to move transportation equipment over long distances, it is not possible to wait until a customer request is known before moving the equipment. As a result, it is necessary to move equipment to serve demands before they are known. In actual applications, there are other sources of randomness such as transit times and equipment failures.

Fleet management problems in practice are quite rich, and it is helpful to focus on a particular application to provide a motivating context. In this chapter we use the classic problem of car distribution in railroads. The problem of optimizing the flows of empty freight cars for railroads is typically formulated as a textbook transportation problem. There are supplies of empty cars, and customers placing orders for these cars. Standard practice is to model the cost of an assigment as the time required to get from the location of a particular car supply to a customer demand. Once we have the supplies, demands and costs, we just throw it into our favorite LP package or network solver and we are done!

This is fairly close to what is being done in engineering practice today. At least two

major railroads in North America are solving their car distribution problem with this basic model. The advantage of the model is that it is easy to understand, the data requirements are straightforward, and it is easy to solve. The question is, does it work?

The answer is, sort of. Any model is an approximation, and the car distribution models used in practice today involve many approximations, but they can still add considerable value compared to manual fleet management. Of particular interest in this chapter, models today either ignore customer demands in the future or resort to deterministic forecasts. By contrast, interviews with planners reveal different strategies to handle the uncertainties of rail operations.

Actual rail networks are complex, noisy operations, and the management of cars has to be done in a way that recognizes all the sources of uncertainty. Surprisingly, the demands of customers for rail cars is not generally the most significant source of noise, primarily because they are in the habit of booking their orders well in advance. Furthermore, many of the largest customers are very predictable (although some are notoriously unpredictable). Perhaps the most annoying source of uncertainty for rail is the transit times, which might range between two and eight days between a pair of cities.

Another source of uncertainty is the supply of new empty cars. Cars may be sent "off line" to another railroad. The other railroad may then return the car empty, but with virtually no advance notice. As a result, empty cars will arrive from another railroad in a highly unpredictable manner. There is also the problem of cars breaking down or being judged unacceptable for service by the customer (typically because they are not clean enough).

Car distribution managers learn to deal with all this uncertainty by maintaining extra inventories of cars, allowing them to quickly substitute alternative cars when events do not proceed as planned. But how many extra cars should be provided, and what type? And where should they be stored? And how do we decide when there are simply too many cars, and there is an opportunity to reduce the fleet? Furthermore, there are some periods of the year when demands are higher than normal, which means that inventories will have to be tightened.

In addition to the challenge of planning inventories, deterministic models have an annoying property that can produce serious practical problems. When a car is emptied, it is best when it can be immediately moved to the area where it will eventually be needed. If not, it has to be brought into a local yard, stored, and then finally moved in response to a customer demand (typically a fairly long distance). It is much better to move the car right away to the general area where it will be needed, and then move the car a much shorter distance when the customer demand actually arises. Deterministic models can use a forecast of future demands, but what happens when the total supply of cars is greater than the point forecast of future car orders? A deterministic model is not able to move the car to a location where it *might* be needed.

Current advances in empty car distribution are focusing on improving the model of the physical process: the movement of trains, train capacities, yards, travel times, and times through yards. But these models completely ignore the modeling of the evolution of in-

formation. Stochastic programming provides a framework for modeling the evolution of information much more accurately than is done with current technologies.

2 The car distribution problem

We begin with a general description of the car distribution problem. Section 2.1 describes the physical processes that govern car distribution, followed by section 2.2 which describes the information processes.

2.1 The physical process

Rail operations can be hopelessly complex, but for the purpose of our model, we are going to represent them in a very simple way. We start when a car becomes empty and available at a shipper's location. The shipper notifies the railroad which then schedules a *local* train to come out and pick up the available cars and bring it back to a *regional yard*. It is easiest for the railroad if the car already has an assignment to a final destination, since this allows the railroad to put the car on the right tracks at the regional yard so that it can move on the correct outbound train.

Cars are moved in *blocks* of cars that share a common origin and destination. However, it is not always possible to build a block of cars that goes all the way to the final destination of the car. Instead, it may be necessary to build blocks of cars to intermediate locations called *classification yards*. Here, blocks can be broken up and divided on multiple outbound tracks, each representing a new block of cars with a common destination. Needless to say, breaking up a block of cars at a classification yard takes time and adds a considerable amount of uncertainty. On the other hand, they offer an important opportunity: the ability to make a decision.

It can take a week or more to move a car from origin to destination. Needless to say, there is a lot of information arriving during this time, and it may be the case that the railroad would like to make a new decision about the car. This can only happen at classification yards. After moving through one or more classification yards, the car finally arrives at a *regional depot*, from which the car may be delivered to a number of customers in the region.

The network is depicted in Figure 1. There are four types of decision points: the customer location when the car first becomes empty, the regional depot where the car is first placed when pulled from the customer, intermediate classification yards, and the regional depot at the destination. Once a car is placed at a destination regional depot, it should not be moved again to another regional depot. However, a car arriving at a classification yard can, in principle, be sent to any other classification yard, or any regional depot.

One of the most powerful strategies that are used by railroads to handle uncertainty is substitution. Substitution occurs across three primary dimensions:

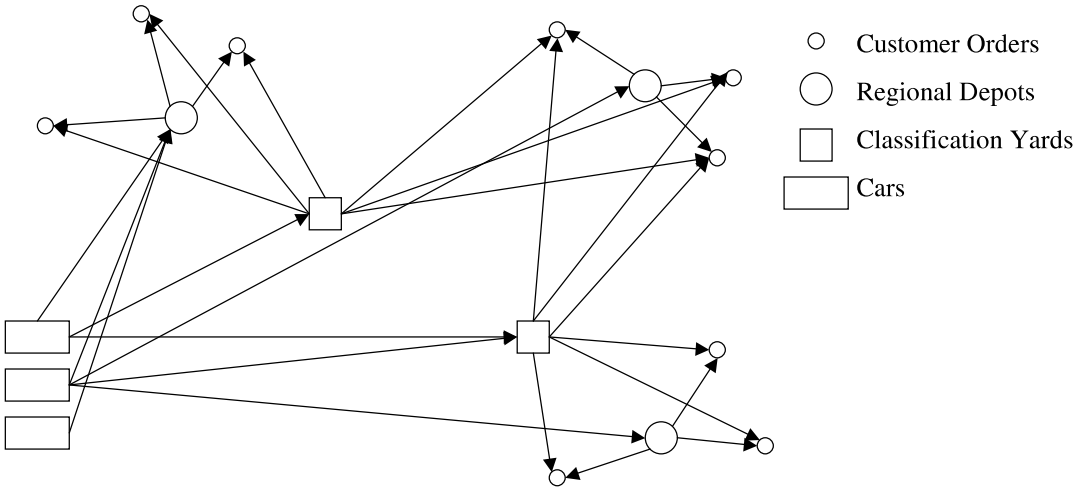

Figure 1: Car distribution through classification yards

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.

Rail cars come in a variety of types. There are about 30 car *groups*, which include cars such as box cars, gondolas, coal cars, tanker cars, and so on. Within a group, there could be anywhere from five or six to as many as two dozen specific types of cars. As a general rule, a customer will require a car from a particular group, and may even require a car of a particular type within a group. Substitution across groups is typically not possible.

Temporal substitution, which also enables geographic substitution, is governed in large part by the characteristics of the customer order. Some car orders need to be satisfied on a particular day, while many can be satisfied any day within a week. Customers that offer within-week flexibility introduce other constraints. A customer may need 25 cars over the course of the week, but is unable to take more than 10 cars on any given day, and needs at least two or three cars each day to keep the warehouse busy.

2.2 The informational process

Our car distribution problem evolves as a result of flows of exogenous information processes and decisions (which might be thought of as endogenous information processes). There are five classes of exogenous information processes:

1) Car orders - Customers call in orders for cars, typically the week before when they are needed. The car order does not include the destination of the order.

2) Order destination - The destination of the order is not revealed until after the car is loaded.

3) Empty cars - Empty cars become available from four potential sources: cars being emptied by shippers, empty cars coming on-line from other railroads, cars that have just been cleaned or repaired, and new cars that have just been purchased or leased.

4) Transit times - As a car progresses through the network, we learn the time required for specific steps (after they are completed).

5) Updates to the status of a car - Cars break down ("bad order" in the language of railroads) or are judged (typically by the customer) to be not clean enough.

The flows of cars are, of course, affected by the decisions that are made.

1) Move car to a location - An empty car may be moved to a regional depot or an intermediate classification yard.

2) Assign to a customer order - Here we make the explicit assignment of a car to a specific customer order.

3) Clean or repair a car - This produces a change in the status of the car.

4) Switch pools - Many cars belong to shipper pools which may be adjusted from time to time.

5) Buy/sell/lease decisions - These decisions affect the overall size of the fleet.

Our presentation will consider only the first two classes, although classes 3 and 4 represent trivial extensions.

It is useful to look at some actual data streams to gain an appreciation of the amount of noise that railroads have to deal with. Figure 2 shows an actual set of demands for a particular type of car at a single location. Also shown is a smoothed estimate of the demands (a point forecast) as well as 10^{th} and 90^{th} percentiles. This figure demonstrates the tremendous amount of noise that railroads have to deal with. What they do not show is the timing of this informatios. Car demands are typically known the week before they have to be filled. By contrast, empty cars are known only as they become available. From this perspective, the noise in the empty car supplies is more difficult to deal with.

Perhaps the most problematic issue for railroads (and their customers) is the uncertainty in the transit times. Data from a railroad demonstrated that a transit time of six days can easily range from four to 10 days. Shorter times can arise when additional trains are scheduled that provide faster than normal service. Longer times reflect delays in classification yards when cars are held for various reasons.

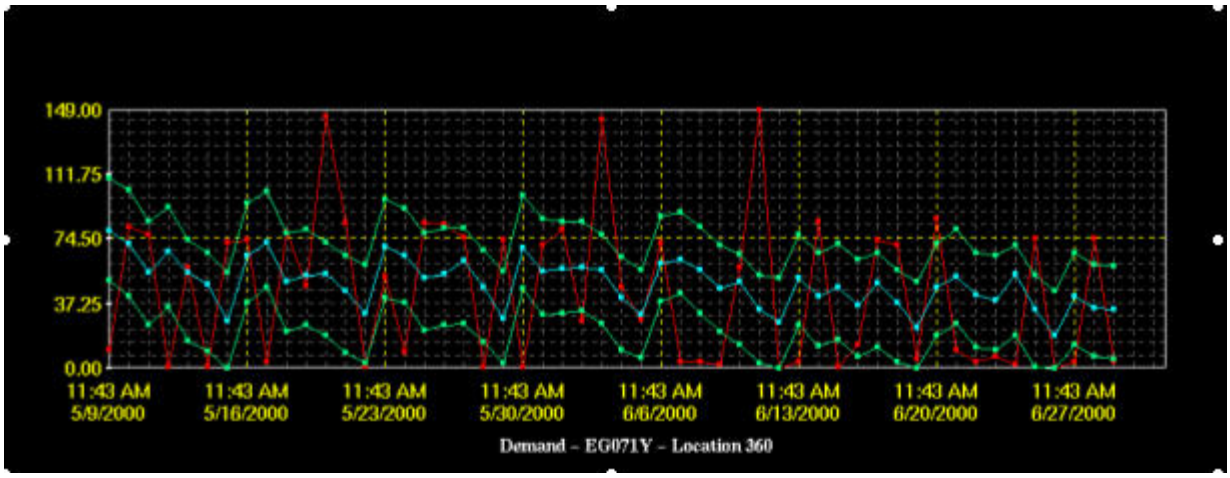

Figure 2: Actual vs. predicted forecasts of future demands, showing the 10^{th} and 90^{th} percentiles

3 A car distribution model

We now translate our physical problem into a mathematical model. Section 3.1 provides our notation. Section 3.2 provides a simple model that is commonly used by railroads today.

3.1 Notation

Our notation is organized along the following elements: the network, the resources being managed, the decisions being made, exogenous information processes, system dynamics, and the objective function.

The network

We assume that cars move between three types of locations:

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

A car that has just become available at a customer can be assigned to any location, as can a car at a classification yard. Cars at regional depots fall in two categories: those that have been pulled in by a local from a customer that just emptied the car, and those that have been moved empty to the depot with the idea of serving a customer in that region. Cars in the second group are supposed to be assigned only to customers in that region. As a result, the decision to move a car empty to a regional depot should be planned with the expectation that the car cannot later be moved to yet another regional depot (or to a customer not served by the first regional depot).

The resources classes

It is customary to model car distribution problems as multicommodity flow problems. This implies characterizing a car by a state (typically its location) and a class (the car type). Our work with real problems has shown that real cars, in practice, are characterized by a more complex set of attributes. For this reason, it is easier to characterize a car by a vector of attributes. Let:

$a =$ The vector of attributes characterizing a car.

$\mathcal{A} =$ The set of attributes of the cars.

$R^c_{t,at'} =$ 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.

$R^c_{t,t'} = (R^c_{t,at'})_{a\in\mathcal{A}}$

$R^c_t = (R^c_{t,t'})_{t'\in\mathcal{T}}$

One of the elements of an attribute vector a would be the location of a car or order. Rather than index the location explicitly, we simply make it one of the attributes. This notation simplifies the process of capturing other attributes of a car such as its maintenance status, cleanliness, or assignment to a shipper pool. However, geographical location is an especially important attribute, and sometimes we need a mechanism for referring to all cars at a particular location. For this we use:

$\mathcal{A}_i =$ The subset of attributes where the location element of a is equal to i.

A more subtle element is the modeling of the status of a resource while it is moving between two locations. If we assume transit times are deterministic, then it is easy to let one attribute of a be the destination of a car, and a second attribute be the time that it is expected to arrive there. Since transit times are neither deterministic nor memoryless, it is necessary to model the origin of the car, and the time that it was dispatched. With this information, it is straightforward to derive the distribution of the time remaining until it arrives at the destination.

Although it is not conventional to think of a customer order as a "resource," mathematically, they are modeled the same way. We let:

$b =$ The vector of attributes characterizing an order.

$\mathcal{B} =$ 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).

$R^o_{t,bt'} =$ The vector of car orders with attribute $b \in \mathcal{B}$ that we know about at time t which are needed at time t'. $R^o_{0,bt'}$ is the set of orders that we know about now.

$R^o_{t,t'} = (R^o_{t,bt'})_{b\in\mathcal{B}}$

$R_t^o = (R_{t,t'}^o)_{t' \in \mathcal{T}}$

The resource vector $R_t = (R_t^c, R_t^o)$ captures what is *known* at time t. The vector $R_{tt'} = (R_{tt'}^c, R_{tt'}^o)$ captures what is known at time t and *actionable* at time t'. The difference between knowability and actionability is often overlooked in modeling. Deterministic models assume everything is known now (typically $t = 0$). Stochastic models typically assume that resources are actionable as soon as they are knowable. For the car distribution problem, the difference between knowable and actionable times is a major modeling element which cannot be overlooked.

Decisions

The decision classes are:

$\mathcal{D}^c =$ 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 $\mathcal{I}^c$).

$\mathcal{D}_t^o =$ The decision to assign a car to a specific order. In this case, $\mathcal{D}_t^o$ can be either the set of all orders available at time t (every element of $\mathcal{D}_t^o$ is a specific order), or the set of all order types (basically, $\mathcal{B}$). We use the latter interpretation, which also means that we can drop the index t (since the set of order types is static). If $d \in \mathcal{D}^o$ is the decision to assign a car type, we let $b_d \in \mathcal{B}$ 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").

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 (for example, in some industries it is possible to contract out a customer request, but we do not have this option). Of these, decisions in $\mathcal{D}^o$ are constrained by the number of orders that are actually available at time t (this constraint is captured in the resource vector R_t^o). Often, the set of decisions depends on the attributes of the resource being acted on. Let:

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

Our decisions are represented using:

$x_{tad} =$ The number of times that we act on a car 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.

Our notation has been chosen to capture the informational dynamics of the problem. For example, standard notation in transportation is 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). Our decision variable is indexed by what is known when a decision is made, and we use the $\delta()$ function to capture the outcome of the decision.

Our sequencing of the subscripts t, a and d is also chosen for a reason. If x_{tad} is a particular decision, it is common to sum over all the decisions d that might act on a resource with attribute a. We then have to sum over all the resources with attribute a at time t. Finally, we might sum the activities over all time periods. Thus, there is a natural hierarchy among the indices t, a and d which we reflect in how we have ordered them.

For the moment, we do not address the question of how a decision is made, but we represent our decision function using:

$X^\pi(R_t) =$ The decision function which returns a decision vector $x_t \in \mathcal{X}_t$, where $\mathcal{X}_t$ is our feasible region and R_t is our general resource vector at time t. We assume that there is a family of decision functions $(X^\pi)_{\pi\in\Pi}$ from which we have to choose.

Exogenous information processes

Our system is driven by a series of exogenous information processes. These can be divided between two broad classes: updates to the "resources" being managed (where we include both cars and car orders), and updates to the parameters that drive our system (the modify function). We model updates to the resources using:

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

$\hat{R}^o_t = (\hat{R}^o_{tt'})_{t'\geq t}$.

We would define similar vectors $\hat{R}^c_{tt'}$ and $\hat{R}^c_t$ for new cars becoming empty.

The second important information process for our model are the transit times. The simplest model is to assume that transit times become known immediately after a decision is made. We might define, then:

$\hat{\tau}_{tad} =$ The actual transit time resulting from a decision d acting on a car with attribute a at time t.

$\hat{\tau}_t = (\hat{\tau}_{tad})_{a\in\mathcal{A},d\in\mathcal{D}}$

A final source of exogenous information arises when the expected state of the car at the destination, which we have represented by the terminal attribute function $a^M(t,a,d)$, is different than what we expected. We can model this by letting $\hat{a}^M_{tad}$ be the random variable which determines the final attribute of a car after it is acted on. However, this degree of realism is beyond the scope of our presentation.

Let W_t be our general variable representing the information arriving at time t. Then:

$$W_t = (\hat{R}^o_t, \hat{R}^c_t, \hat{\tau}_t)$$

is our exogenous information process. Let ω be a sample realization of $(W_t)_{t\in\mathcal{T}}$, and let Ω be the set of potential outcomes. Further let $\mathcal{F}$ be the $\sigma-$algebra generated by $(W_t)_{t\in\mathcal{T}}$, and let $\mathcal{F}_t$ be the sub-$\sigma-$algebra generated by $(W_{t'})^t_{t=0}$, where $\mathcal{F}_t$ forms a filtration. If $\mathcal{P}$ is a probability measure on Ω, then we can let $(\Omega, \mathcal{F}, \mathcal{P})$ be our probability space.

Throughout this chapter, we use the index t to represent the information concept of a variable or function. So, a variable R_t or a function Q_t is assumed to be $\mathcal{F}_t$-measurable.

System dynamics

We assume that the effects of a decision are captured by the *modify function* which can be represented using:

$$M(t,a,d) \rightarrow (a', c, \tau) \tag{1}$$

where d is a decision acting on a car with attribute a at time t, producing a car 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 $(a^M(t,a,d), c^M(t,a,d), \tau^M(t,a,d))$. 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 $\tau_{tad} = \tau^M(t,a,d)$. It is not generally the case that $(a^M(t,a,d), c^M(t,a,d), \tau^M(t,a,d))$ are deterministic functions of (t,a,d). This is particularly true of the transit time $\tau^M(t,a,d)$.

For algebraic purposes it is useful to define:

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

The dynamics of our car inventories using:

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

3.2 A simple car distribution model

We can quickly illustrate the basic deterministic car distribution models that are used in practice. A myopic model would be given by:

$$\max_x \sum_{a\in\mathcal{A}}\sum_{d\in\mathcal{D}} c_{0ad}x_{0ad} \tag{3}$$

subject to:

$$\sum_{d\in\mathcal{D}} x_{0ad} = R^c_{0a} \quad a\in\mathcal{A} \tag{4}$$

$$\sum_{a\in\mathcal{A}} x_{0ad} \leq R^o_{0b_d} \quad d\in\mathcal{D}^o \tag{5}$$

$$x_{0ad} \in Z_+. \tag{6}$$

Keep in mind that for every element d in the set of order decisions $\mathcal{D}^o$ corresponds to an order *type* with attribute b_d.

A model that incorporates point forecasts of future car orders is a minor variation. Let $\bar{R}^o_{tb}$ be a point forecast of the car orders that are made (that is, become known) at time t of type b. A deterministic model that incorporates this forecast is given by:

$$\max_x \sum_{a\in\mathcal{A}}\sum_{d\in\mathcal{D}} c_{0ad}x_{0ad} \tag{7}$$

subject to:

$$\sum_{d\in\mathcal{D}} x_{0ad} = R^c_{0a} \quad a\in\mathcal{A} \tag{8}$$

$$\sum_{a\in\mathcal{A}} x_{0ad} \leq R^o_{ob_d} + \sum_{t\in\mathcal{T}^{ph}\setminus 0} \bar{R}^o_{tb_d} \quad d\in\mathcal{D}^o \tag{9}$$

$$x_{0ad} \in Z_+. \tag{10}$$

Equation (9) includes demands that are known now (R^o_{0b}) and all orders that are forecasted to become known within the planning horizon. We have not included point forecasts of new empty cars becoming available. While this is easy to do, it is not common practice. As a result, this model has a bias wherein it includes future orders but not future cars.

These simple models are attractive in part because they are easy to understand, easy to formulate and solve, and appear to capture the way railroads actually make decisions. For example, it is common to assign cars available now to known (and sometimes even forecasted) orders, ignoring cars that may become available in the future. Conversations with actual

planners, however, reveal that they will look in the aggregate at total orders that are expected to arise, and total cars that are expected to become available, to determine whether they seem to be in a surplus or deficit situation. Often, a car may become available for which there is not a pending order. Rail operations work best if the car is assigned to a destination as soon as it becomes empty, forcing planners to move cars to locations where there appears to be a good likelihood of being used.

The sections which follow introduce stochastic formulations of this model, beginning in section 4 with a two-stage stochastic program, followed by section 5 which reviews algorithms for two-stage problems. Section 6 presents a multistage stochastic programming formulation, which can be solved as sequences of two-stage problems.

4 A two-stage model

The management of rail cars over time is a multistage resource allocation problem. But it is not unreasonable to formulate it as a two stage problem. The justification arises because of the nature of the car distribution problem. In any given week, a railroad has to decide how to move cars one week to meet demands that will arise in the next week. Transit times are quite large, and it will generally take several days to a week or more to move a car. For this reason, decisions to assign a car to an order within the same week are typically restricted to cars that are already at the regional depot and which now just have to be pulled by a local train from the regional depot to a customer within the region.

Reflecting these long transit times, it is common industry practice for customers to place an order one week that needs to be met the following week. This means that by Friday morning of a week, over 90 percent of the orders for the next week are known. However, most car orders are made on Wednesday and Thursday of a week. This means that decisions to move cars early in the week still have to be made without much of the information that will become available the following week. As a result, it is often necessary to make a decision to move a car on Monday or Tuesday, where the actual demand will become known only after the decision is made.

Once a car has been assigned to an order, the transit time typically takes the car past the end of a two week horizon. We are typically not concerned with where the car is ultimately going, since our problem right now is simply to cover the order. Thus, all the activity (including all the costs and revenues) that we are concerned with right now occurs within the next two weeks, during which time a car will, with rare exception, be able to cover at most one demand. For this reason, a two-stage formulation is a reasonable starting point (and helps to lay the foundation for a multistage model).

$$\max_{x_0} \left\{ c_0 x_0 + \bar{Q}(R_0^{c,x}) \right\} \tag{11}$$

subject to:

$$\sum_{d\in\mathcal{D}} x_{0ad} = R^c_{0a} \quad a \in \mathcal{A} \tag{12}$$

$$\sum_{a\in\mathcal{A}}\sum_{d\in\mathcal{D}_a} x_{0ad}\delta_{1a'}(0,a,d) - R^{c,x}_{0a'} = 0 \quad a' \in \mathcal{A} \tag{13}$$

$$R^{c,x}_{0a'} + \hat{R}^c_{1a'} - R^c_{1a'} = 0 \tag{14}$$

$$\sum_{a\in\mathcal{A}} x_{0ad} \leq R^o_{0b_d} \quad d \in \mathcal{D}^o \tag{15}$$

$$x_{0ad} \in Z_+. \tag{16}$$

$R^{c,x}_{0a'}$ represents the cars that will have attribute a' (for stage 1) as a result of decisions made in stage 0. The variable is indexed by time 0 because it is a function of the information available in time 0. This notation may seem a bit awkward in this setting, but it becomes more important in the context of multistage problems (and especially when we have to deal with multiperiod travel times).

The second stage problem consists of finding:

$$\bar{Q}(R^{c,x}_0) = EQ(R^c_1, R^o_1) \tag{17}$$

The conditional second stage function is given by:

$$Q(R^c_1, R^o_1(\omega)) = max_{x_1(\omega)} \sum_{a\in\mathcal{A}}\sum_{d\in\mathcal{D}^o_a} c_{1ad}x_{1ad}(\omega) \tag{18}$$

subject to:

$$\sum_{d\in\mathcal{D}^c_r} x_{1ad}(\omega) + x_{1d^\phi a}(\omega) = R^c_{0a} \quad \forall a \in \mathcal{A} \tag{19}$$

$$x_{1ad}(\omega) \leq R^o_{1b_d}(\omega) \quad \forall d \in \mathcal{D}_a,\ a \in \mathcal{A} \tag{20}$$

$$x_{1ad}(\omega) \geq 0 \quad \forall d \in \mathcal{D}_a, a \in \mathcal{A}. \tag{21}$$

This is a classical two-stage stochastic program with network recourse, with one twist. In practical problems, the space $\mathcal{A}$ may be too large to enumerate.

5 Solution algorithms for two-stage problems

There are a variety of algorithms for two-stage problems with network recourse. Our choice of algorithms has to be guided by several factors. First, we would like integer solutions. Second, the problem may be quite large, especially if the attribute vector a has enough

dimensions that the space $\mathcal{A}$ is too large to enumerate. These characteristics effectively eliminate scenario methods (which produces problems that are too large) as well as stochastic linearization and Benders decomposition (which produce highly fractional solutions). What remains are methods that approximate the second stage recourse function in a way that retains as much of the inherent network structure as possible.

We present two methods that have shown promise for this problem class. Both are techniques for deriving separable, nonlinear approximations of the value function. Since we are interested in integer solutions, we use piecewise linear approximations with breakpoints occurring only for integer values of supplies. Section 5.1 describes how to estimate these functions using auxiliary function methods. Section 5.2 describes a relatively new class of structured adaptive function methods that we call "SAFE" algorithms.

5.1 Auxiliary function methods

We are not able to estimate the function $\bar{Q}(R)$ in equation (17). However, we are able to solve equations (18)-(21) fairly easily for a single, sample realization. Let $\tilde{q}_1$ be the vector of dual variables for equation (19). We could use these duals, which are stochastic gradients of $\bar{Q}(R_0^c)$, to create a linear approximation of the value function by first smoothing the gradients:

$$\hat{q}^n = (1-\alpha^n)\hat{q}^{n-1} + \alpha^n \tilde{q}^n$$

We could then solve:

$$x_0^n = \arg\max_{x_0} c_0 x_0 + \hat{q}^n R_0^{c,x}(x_0).$$

This sort of solution procedure, however, would be very unstable and would not converge. It is necessary to introduce the smoothing step of the form:

$$\bar{x}^n = (1-\alpha^n)\bar{x}^{n-1} + \alpha^n x_0^n.$$

Unfortunately, a smoothing step such as this would destroy the integrality of the solution. The better alternative is to use an auxiliary function. The SHAPE algorithm starts with an initial nonlinear approximation $\hat{Q}^0(R)$ which is then updated using the strategy:

$$\hat{Q}^n(R) = \hat{Q}^{n-1}(R) + \alpha^n \left(\tilde{q}^n - \nabla \hat{Q}^{n-1}(R^n)\right) R. \tag{22}$$

We would then solve sequences of problems of the form:

$$x_0^n = \arg\max_{x_0} c_0 x_0 + \hat{Q}^{n-1}(R_0^{c,x}(x_0)) \tag{23}$$

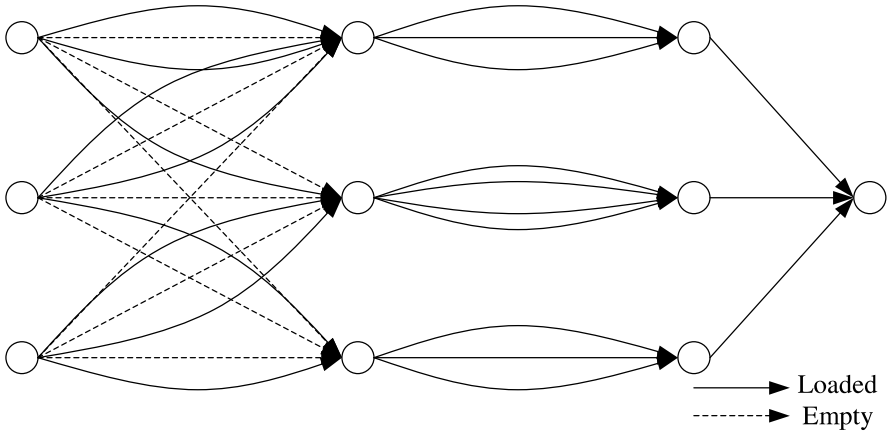

Figure 3: Network structure of a one-period single commodity problem

We then use $R^n = R_0^{c,x}(x_0)$ in equation (22). The approximation $\hat{Q}^0(R)$ should be concave (since the expected recourse function is also concave). It is also natural to choose separable approximations. Some choices are:

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

This algorithm is provably convergent if the expected recourse function $\bar{Q}(R_0^{c,x})$ (and the approximation $\hat{Q}^0(R)$) are continuously differentiable. However, it can be applied approximately to nondifferentiable functions. For these problems, we can choose $\hat{Q}^0(R)$ so that it is piecewise linear, concave, and separable. For example, we could use piecewise linear versions of equations (24)-(24). When this choice of function is made, equation (23) involves solving sequences of networks that are depicted in Figure 3. These are network subproblems and naturally produce integer solutions (assuming that the piecewise linear approximations have breaks at integer values of the supplies).

The SHAPE algorithm yields sequences of computationally tractable approximations that are very easy to solve, and which also naturally produce integer solutions. However, the shape of the recourse function is fixed by the choice of the initial approximation (such as equations (24)-(24)). The updating equation will tilt these functions, but does not change their basic shape. This means that the quality of the solution produced by the algorithm may be fairly dependent on the choice of the parameters that characterize $\hat{Q}^0(R)$. The next section presents an algorithm that overcomes this limitation.

5.2 Structured, adaptive function estimators

Structured, adaptive function estimators, or "SAFE" algorithms, produce piecewise linear, concave, separable approximations of the recourse function by using sample gradients to

update portions of the recourse function. No assumption is made about the shape of the recourse function, other than that it is piecewise linear and concave. Furthermore, it is necessary to maintain concavity at every iteration. SAFE algorithms still involve solving sequences of problems using the strategy given in equation (23), and produces problems with the same mathematical structure. The only difference is the calculation of the approximation $\hat{Q}^n(R)$.

SAFE algorithms use stochastic gradients to update estimates of a piecewise linear function while maintaining concavity after each update. There are several schemes for doing this, although here we describe a simple projection method that appears to work very well, and also has proven convergence properties.

Let $Q(r), r \in \Re$, be a piecewise linear, concave function where r is a scalar argument, and let

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

be the right derivative of $Q(R)$ at $R = r$. We can write $Q(r)$ in terms of its derivatives using:

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

For our problems, we can always let $Q(0) = 0$. Let $\hat{q}_r^n$ be an estimate of q_r at iteration n, giving us the functional approximation:

$$\hat{Q}^n(R) = \sum_{r=0}^{R-1} \hat{q}_r^n.$$

Let $\tilde{q}^n$ be a stochastic gradient of Q at iteration n, and assume the gradients have the property that $E[\tilde{q}_r^n] = q_r$. Assume that at iteration n we sample $r = R^n(\omega)$. If $\bar{q}_r^n$ is our estimate of the slope for $R = r$, then we can estimate the slopes using the simple updating equation:

$$\bar{q}_r^n = \begin{cases} (1-\alpha^n)\bar{q}^{n-1} + \alpha^n \tilde{q}^n & \text{if } r = R^n(\omega) \\ \bar{q}_r^{n-1} & \text{Otherwise.} \end{cases} \tag{24}$$

If we 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 $q_0 \geq q_1 \geq \ldots \geq q_r$. Equation (24) would not maintain this relationship between the slopes. A loss of concavity at an intermediate iteration would make it very difficult to solve equation (23) (and obtain proper dual variables). Concavity is automatically maintained in equation (22) since we are updating a concave approximation with a linear updating term.

We can maintain concavity by using a simple projection algorithm.

$$\hat{q}^n = \Pi_{\mathcal{Q}}(\bar{q}^n) \tag{25}$$

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

$$\max_{\hat{q}^n} \|\hat{q}^n - \bar{q}^n\|^2 \tag{26}$$

subject to:

$$\hat{q}^n_{r+1} - \hat{q}^n_r \leq 0. \tag{27}$$

The projection is easily solved. Assume that after the update (equation (24)), we have an instance where $\bar{q}^n_{r-1} < \bar{q}^n_r$. Let $\bar{r} = argmin_{r'<r}\{\bar{q}^n_{r'} < \bar{q}^n_r\}$ be the smallest index such that $\bar{q}^n_{\bar{r}} < \bar{q}^n_r$. Our projection is found by finding 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

$$\hat{q}^n_{r'} = \begin{cases} \bar{q}^n_{[\bar{r},r]} & \text{if } \bar{r} \geq r' \geq r \\ \bar{q}^n_r & \text{Otherwise} \end{cases}.$$

5.3 Extension to large attribute spaces

A special challenge in resource allocation problems arises when the attribute vector a has multiple dimensions (even as few as five or six). In these cases, it may not be possible to enumerate the attribute space $\mathcal{A}$. This means that we may need an estimate of the value of a resource with attribute a' but we do not have an approximation $\hat{Q}_{a'}$. An effective strategy can be to produce approximations $\hat{Q}^{(n)}_{a^{(n)}}$ where $a^{(n)}$ is the n^{th} aggregation of the attribute vector a. If $n = 0$ represents the most disaggregate level, then $\hat{Q}^{(n)}_{a^{(n)}}$ for increasing values of n will produce lower statistical error and higher structural error.

5.4 Experimental work for two-stage problems

Two stage stochastic programs in general, and those with network recourse in particular, have been widely studied. Lacking special structure, the most widely cited approach for solving two-stage stochastic programs is based either on scenario methods (which are computationally intractable for our problems) and Benders decomposition. Benders decomposition comes

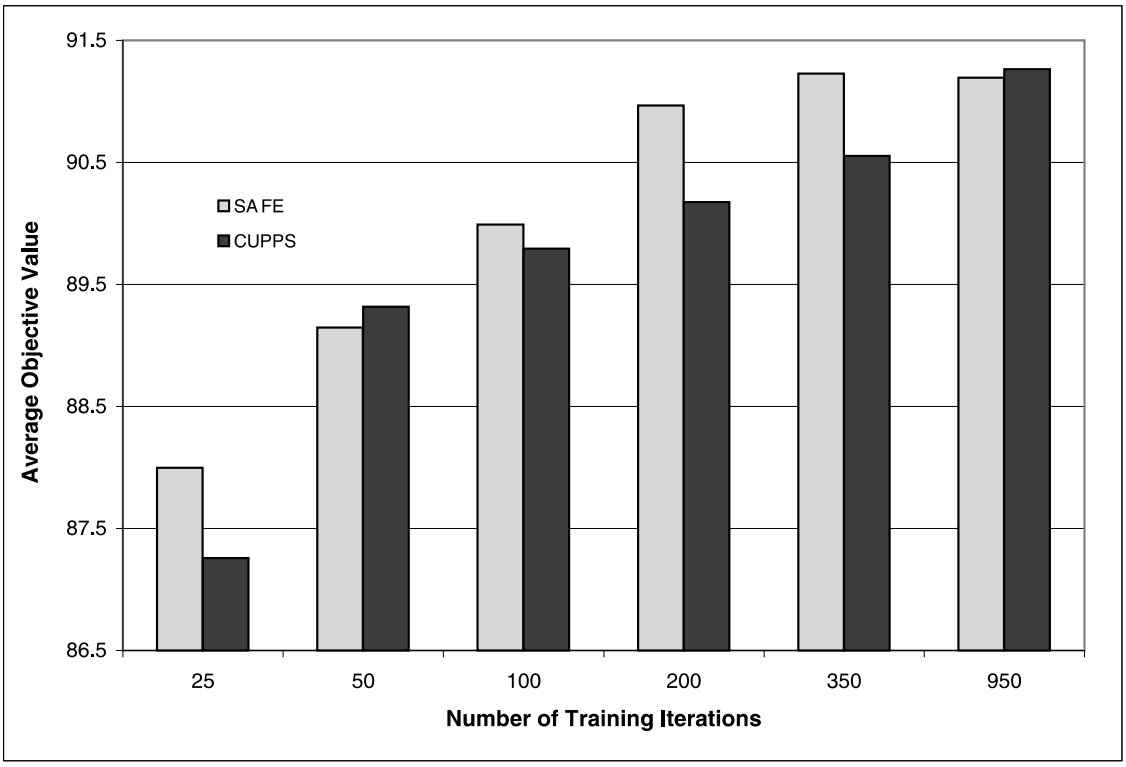

Figure 4: The effect of the number of training iterations on SAFE and CUPPS, illustrating the faster convergence for SAFE

in different flavors, including L-shaped decomposition, stochastic decomposition and the recently proposed CUPPS algorithm. L-shaped decomposition assumes a finite, and relatively small, sample of scenarios which are used to generate cuts. Stochastic decomposition is an iterative algorithm that effectively assumes an infinite set of scenarios, and provides almost sure convergence in the limit. CUPPS requires a finite set of scenarios (there is a particular operation that loops over all scenarios) but the operation is very simple and the method can handle thousands of scenarios without difficulty. Although CUPPS requires somewhat more work per iteration, it has faster convergence, and hence we use it as for our comparisons.

Figure 4 shows the relative performance of SAFE and CUPPS for different numbers of training iterations. This graph suggests generally better solution quality for SAFE over smaller number of iterations. The evaluation was based on a testing sample of 50. With 950 training iterations, the algorithms appear to be equivalent.

Figure 5 compares SAFE and CUPPS on problems with 20, 30, 40 and 90 locations, using 200 training iterations. For the smaller problems, the algorithms appear to be virtually the same, while for the 40 and 90 location problems, SAFE is providing better solutions. The chart shows the performance for *each* of the 50 testing samples. For the smaller datasets, the two algorithms appear to be providing the exact same solution for all 50 samples. For the 90 location dataset, SAFE is providing a slightly better answer on *all* 50 samples. This behavior is somewhat surprising, in part because CUPPS is a provably convergent algorithm, and also because we would expect some noise around the behavior of each method.

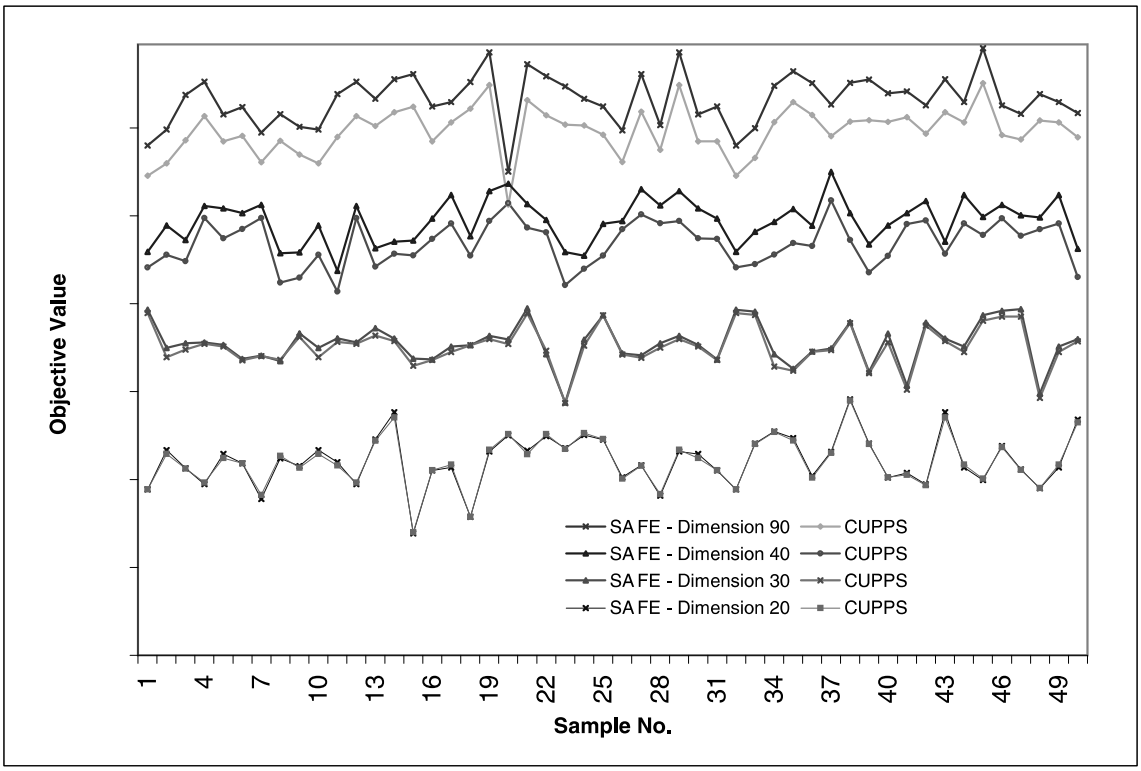

Figure 5: SAFE outperforms CUPPS as the problem size grows for every outcome in the testing dataset

6 Multistage resource allocation problems

Practical, operational models for car distribution can be formulated very approximately as deterministic, single stage models, and quite credibly as two-stage stochastic programs. But there are other planning applications that require looking farther into the future, and for these we need a multistage model. Our approach is to solve the multistage problem as a sequence of two-stage stochastic programs using a dynamic programming framework.

6.1 Formulation

Let:

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

$W_t =$ 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).

We model the "state" of our system using:

$R_t =$ Vector of resources available in period t after new arrivals are added in.

$K_t =$ What is known at time t after the new information W_t has been incorporated. K_t includes R_t plus what we know about parameters that govern the dynamics of the system.

Our process is controlled by the decisions we make:

$x_t =$ Decisions which are made after new information in period t has become known.

$X_t^\pi(R_t) =$ The decision function which returns a feasible decision vector x_t given the information available at time t (contained in K_t).

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

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

The dynamics of the resource state vector for cars are given by equation (2). This equation can be represented more compactly in matrix form using:

$$R_{t+1,t'} = R_{tt'} + A_{tt'}x_t + \hat{R}_{t+1,t'}.$$

The informational dynamics can be written generally as:

$$K_t = U^K(K_{t-1}, W_t).$$

We can now state our optimization problem using:

$$\max_{\pi\in\Pi} E\left\{\sum_{t\in\mathcal{T}} c_t(X_t^\pi(R_t)).\right\}$$

6.2 Our algorithmic strategy

Our solution strategy starts with the standard Bellman equation:

$$Q_t(R_t) = \arg\max_{x_t\in\mathcal{X}_t} c_t x_t + E\left\{Q_{t+1}(R_{t+1}(x_t))|R_t\right\}. \tag{28}$$

We do not know the recourse function exactly, so we replace the expectation on the right hand side with an approximation $\hat{Q}_{t+1}(R_{t+1})$. Assume that we choose a linear approximation:

$$\begin{aligned}\hat{Q}_{t+1}(R_{t+1}) &= \hat{q}_{t+1}\cdot R_{t+1} \\ &= \hat{q}_{t+1}\cdot(R_t + A_t x_t + \hat{R}_{t+1}).\end{aligned} \tag{29}$$

Replacing $Q_{t+1}(R_{t+1})$ with $\hat{Q}_{t+1}(R_{t+1})$ and taking conditional expectations of both sides of (29) gives:

$$\begin{aligned} E\{\hat{Q}_{t+1}(R_{t+1})|R_t\} &= E\{\hat{q}_{t+1} \cdot (R_t + A_t x_t + \hat{R}_{t+1})|R_t\}. \\ &= \hat{q}_{t+1} R_t + \hat{q}_{t+1} A_t x_t + E\{\hat{q}_{t+1}\hat{R}_{t+1}|R_t\}. \end{aligned} \tag{30}$$

The expectation on the right hand side of equation (30) is not a function of x_t and can be dropped. Our decision function as:

$$X_t^\pi(R_t) = \arg\max c_t x_t + \hat{q}_{t+1} A_t x_t$$

Linear approximations are always the easiest to work with. At the same time, they tend to be unstable in practice and typically do not produce good solutions. Convergent algorithms require smoothing on the decision variables, but this will create fractional solutions.

A better approach is to use nonlinear approximations, but this prevents the convenient decomposition that we obtain in equation (30). For this reason, we have to start over with a different state variable. The vector R_t can be called the *complete* state variable because it has all the information that we need to make a decision. But the imbedded expectation causes practical problems. Instead, we define:

$$\begin{aligned} R_{tt'}^x = &\text{ The resource state variable after the decision } x_t \text{ has been implemented, but} \\ &\text{before the new arrivals from period } t+1 \text{ have been added in.} \\ = &\ R_{tt'} + A_{tt'} x_t. \end{aligned}$$

We refer to $R_t^x = (R_{tt'})_{t' \geq t}$ as the *incomplete* state variable because it does not contain the new information (that would arrive during time period $t+1$) that would be needed to make the decision x_{t+1} in the next time period. Our complete state variable can now be written in terms of the incomplete state variable using:

$$R_{t+1,t'} = R_{tt'}^x + \hat{R}_{t+1,t'}.$$

If we formulate our recursion using the incomplete state variable, we would obtain:

$$Q_{t-1}^x(R_{t-1}^x) = E\left\{\arg\max_{x_t \in \mathcal{X}_t} c_t x_t + Q_t^x(R_t^x(x_t))|R_{t-1}^x\right\}. \tag{31}$$

The left hand side of equation (31) is indexed by time period $t-1$ because the right hand side, after taking the expectation, is a function of the information up through time $t-1$.

The expectation in equation (31) is computationally intractable, as it was in equation (28). However, we can take a sample realization and solve

$$\tilde{Q}_t^x(R_{t-1}^x(\omega)) = \arg\max_{x_t \in \mathcal{X}_t(\omega)} c_t x_t + Q_t^x(R_t^x(x_t, \omega)) \tag{32}$$

We finally replace $Q_t^x(R_t^x(x_t))$ with an appropriate approximation $\hat{Q}_t(R_t^x)$ so that equation (32) is still computationally tractable.

$$X_t^{\pi,n}(R_t) = \arg\max_{x_t} c_t x_t + \hat{Q}_t^{x,n-1}(R_t^x(x_t)) \tag{33}$$

subject to:

$$\sum_{d\in\mathcal{D}} x_{tad} = R_{t,at}^c \quad a \in \mathcal{A} \tag{34}$$

$$\sum_{a\in\mathcal{A}} x_{tad} \leq R_{tb_d}^o \quad d \in \mathcal{D}^o \tag{35}$$

$$x_{tad} \in Z_+. \tag{36}$$

Let $R_t^{x,n} = R_t^{x,n}(X_t^{\pi,n}(R_t))$ be the resource vector after we have made decision $X_t^{\pi,n}(R_t)$. Let $\tilde{q}_t^n$ be the dual variable (at iteration n) of equation (34). The process of updating the recourse function approximation, where we will use the same techniques that we did for the two-stage problem, can be represented simply using:

$$\hat{Q}_{t-1}^{x,n} \leftarrow U^Q(\hat{Q}_{t-1}^{x,n-1}, \tilde{q}_t^n, R_t^{x,n}).$$

There are two ways of using this solution approach. The first is a one-pass procedure where the recourse function is updated as we step through time. The steps of this procedure are given in Figure 6. This procedure is the easiest to implement, but can suffer from slow convergence since information is passed backward one time period at a time at each iteration.

More rapid communication is accomplished using a two-pass procedure, described in Figure 7. This procedure can be somewhat harder to implement, since it requires storing information at each time period for computing gradients that are not used until the end of the forward pass. The calculation of gradients from the two-pass procedure is illustratred in Figure 8. As the algorithm steps forward in time, it is necessary to store the basis that was founded when solving $X_t^\pi(R_t)$ (equation (33)). When the subproblem is a pure network, the basis is a flow augmenting path. We have to link these flow augmenting paths through time to compute gradients for the entire simulation.

If we use a two-pass procedure, then our gradients have a nice property. Let:

$$F_t^\pi(R_t, \omega^n) = \sum_{t'=t}^{T} c_t X_t^\pi(R_t(\omega)) \tag{37}$$

be the total contribution of following policy π from time t onward, given that we start at resource state R_t and follow the sample path ω. Then we have the following result:

STEP 0: Initialization:

Initialize $\hat{Q}_t^0, \quad t \in \mathcal{T}$.

Set $n = 0$.

Initialize R_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 (33) 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 (34).

STEP 2b: Update the resource state: $R_{t+1}^n = M_t(R_t^n, x_t^n, W_{t+1}(\omega^n)$.

STEP 2c: Update the value function approximations using $\hat{Q}_t^n \leftarrow U^Q(\hat{Q}_t^{n-1}, \tilde{q}_t^n, R_t^n)$.

STEP 3: Return the policy $X_t^\pi(R_t, \hat{Q}^N)$.

Figure 6: Single pass version of the adaptive dynamic programming algorithm

Theorem 6.1 *Let $\bar{q}_t^n = (\bar{q}_{at}^n)_{a \in \mathcal{A}}$ be the vector of path costs from time t to the end of the horizon, given outcome ω^n and functional approximations $\{\hat{Q}_t^n\}_{t \in \mathcal{T}}$ computed from a backward pass. Then $\bar{q}_t$ satisfies:*

$$F_t^\pi(R_t, \omega^n) - F_t^\pi(R_t', \omega^n) \geq \bar{q}_t^n \cdot (R_t - R_t')$$

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

6.3 Single commodity problems

When we solve single commodity problems in a multistage setting, we are simply solving sequences of two-stage problems (see Figure 3), so we can use all of our tricks that we learned for the simple two-stage case. There are, of course, a few differences. First, it is best to obtain dual variables from the backward pass of a two-pass algorithm, as illustrated in Figure 7. For a two-stage problem, we obtain dual variables directly from the second stage. For multistage problems, we need to calculate these backward through time.

STEP 0: Initialize $\hat{Q}_t^0, \quad t \in \mathcal{T}$.

Set $n = 0$.

Initialize R_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 (33) 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 (34).

STEP 2b: Compute $R_{t+1}^n = M(R_t^n, x_t^n, W_{t+1}(\omega^n))$

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 $X_t^\pi(R_t, \hat{Q}^N)$.

Figure 7: Two- pass version of the adaptive dynamic programming algorithm

Second, multistage problems in transportation typically encounter the problem of multiperiod travel times. That is, it can take several time periods (and possibly many) to get from one location to another. If these travel times are treated deterministically, then we can handle these through an augmented state variable such as:

$R_{t,at'} =$ The number of resources that, at time t, will have attribute a at time t'.

This is a family of resource variables over the range $t' \geq t$. The representation works as long as travel times are deterministic. For rail applications, this is actually a pretty poor approximation. The most general model retains the history of prior decisions:

$x_{t'ad,t}^h =$ The number of resources which at time $t' \leq t$ had attribute a and were acted on by decision d, and which by time t have not yet completed their trip.

We use the notation $x_{t't}^h$ rather than the resource variable $R_{tt'}$ since we are literally keeping a history of prior decisions. This representation is fairly complex, but provides for any probability model for transit times.

6.4 Multicommodity problems

Multicommodity problems are common in rail applications because there are different rail cars that can be substituted. Rail cars are typically organized in groups, with different types of cars arising within a group. A customer may request a specific type of car within a group, while other customers can accept any car within a group. It is with the latter type of customer that substitution occurs.

If we use linear recourse function approximations, the single period problems reduce easily to pure networks. When we use nonlinear approximations, the single period problems produce sequences of (integer) multicommodity network flow problems, depicted in Figure 9. In sharp contrast with multiperiod, multicommodity flow problems, these single period problems are actually relatively easy to solve, and usually produce integer solutions.

6.5 Experimental results for an empty car distribution problem

Separable, piecewise linear value function approximations are especially easy to use for multistage stochastic, dynamic car distribution problems. They provide very near-optimal solutions for two-stage problems, but this performance arises in part because the initial state is deterministic. As the value function approximations converge, the (incomplete) state variable for the second stage also converges.

This same property does not hold true with multistage problems. The vector of resources R_t for a general time $t > 0$ will be random, which means that the separable value functions for stage $t + 1$ must be "good" over a range of outcomes of R_t. It would be comparable to

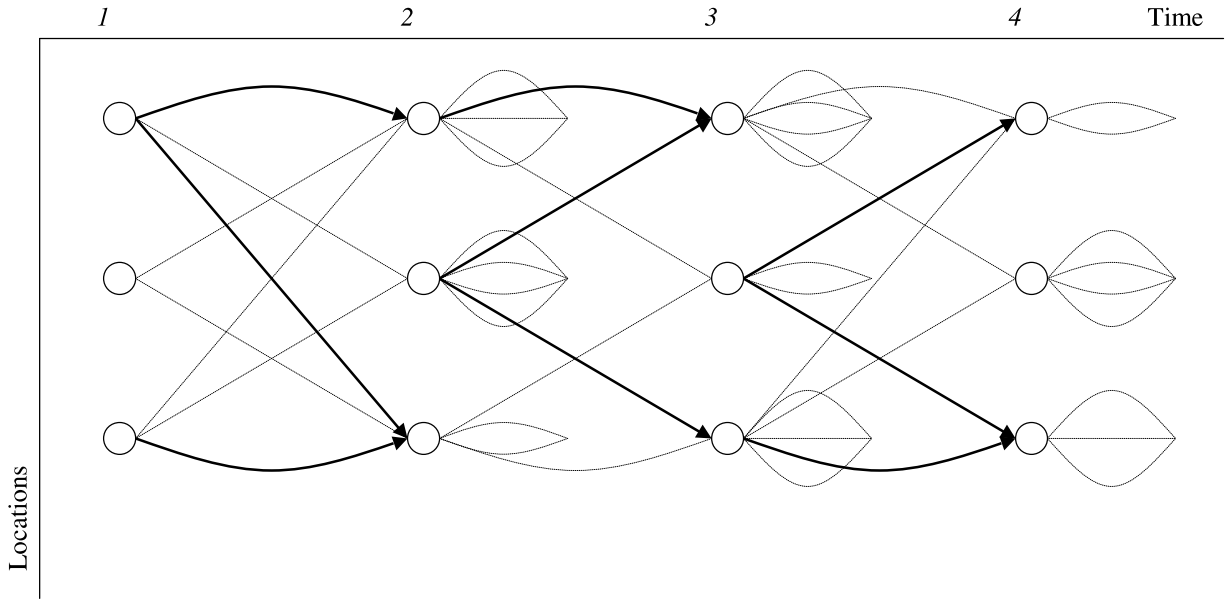

Figure 8: Optimal network basis from the forward pass

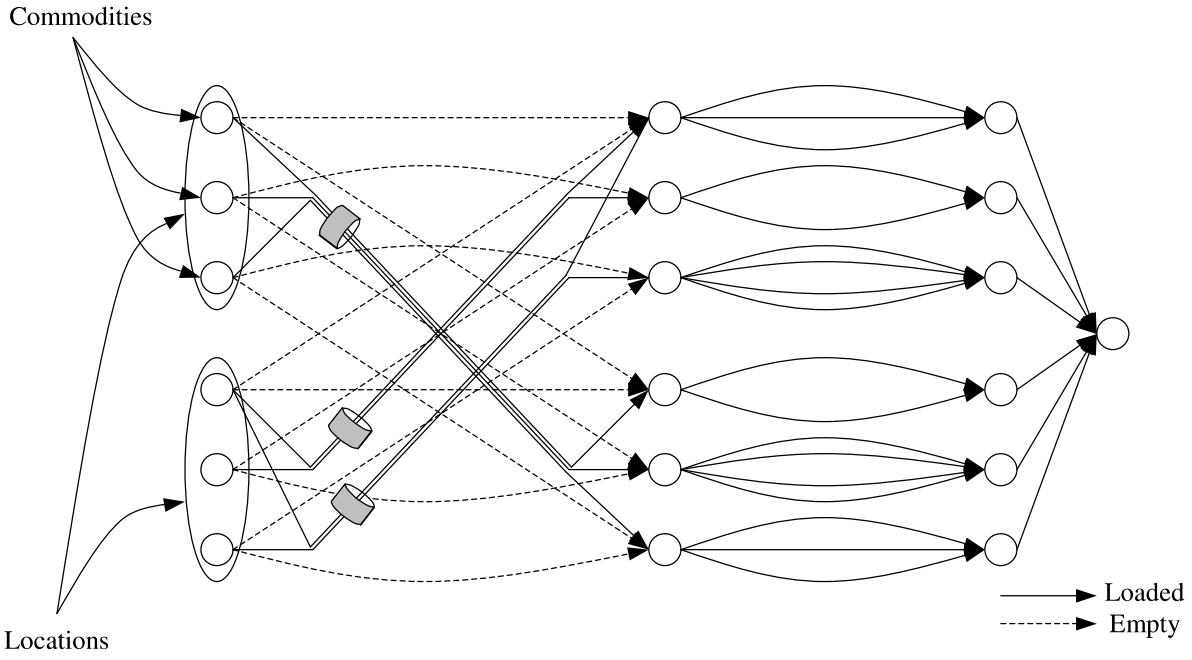

Figure 9: Network structure of a one-period multicommodity problem

	Planning 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 CAVE for second set of deterministic experiments with single-period time windows (network problems)

testing the logic for a two-stage problem where the starting state is random.

There are several ways to test the logic for a multistage problem. The first is to test it on deterministic instances. In practice, it is common for a company to use a snapshot of history (which is deterministic) to see how well the logic would have performed in the past. The research community might challenge this test, but it is a common one in industry. Besides, if the algorithm works well for stochastic problems, it should also work well for deterministic problems.

Table 1 gives a set of results reported in Godfrey & Powell (2002*a*) where a variant of a SAFE algorithm (called the CAVE algorithm, given in Godfrey & Powell (2001)) is tested on a deterministic, dynamic network. The algorithm was tested on 20, 40 and 80 location problems, with 15, 30 and 60 period horizons. The results indicate near-optimal performance. The success for this problem class is closely related to the success for two-stage problems: the value functions steadily converge, meaning the state vector R_t converges to a point, allowing the separable approximations to be locally accurate.

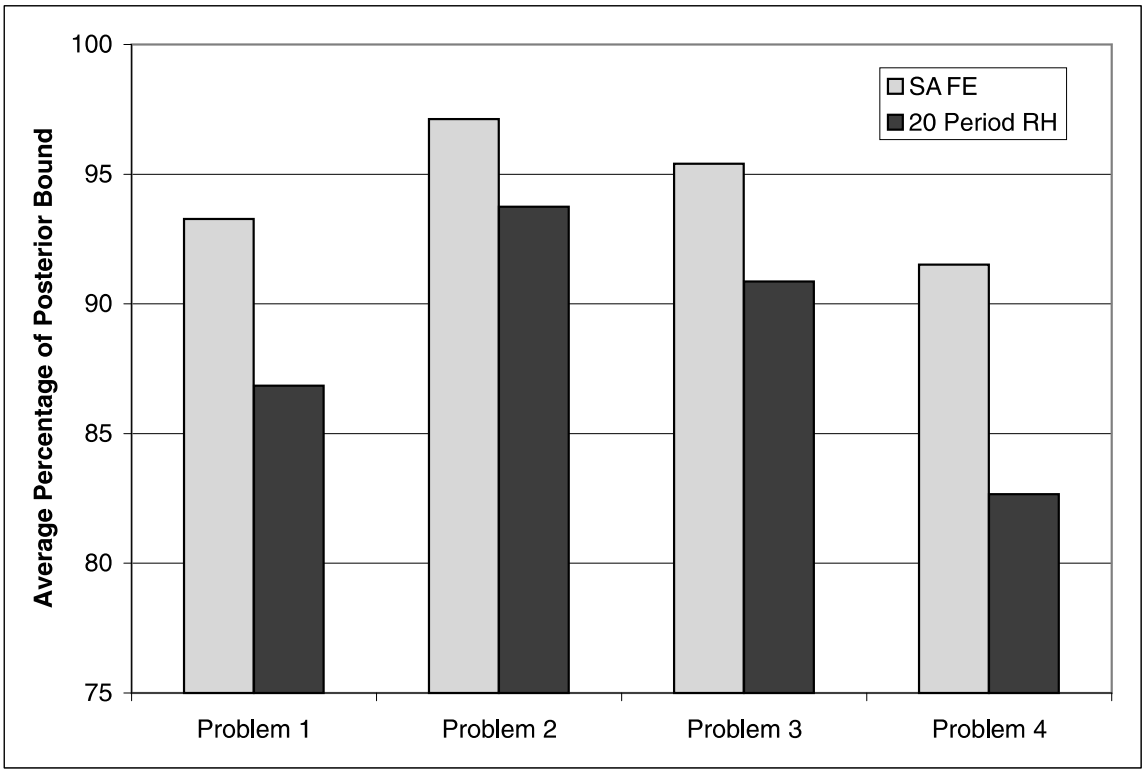

Figure 10: Comparison of SAFE approximations to rolling horizon procedures for 20 location datasets and different substitution matrices

A second set of experiments (from Topaloglu & Powell (2000)) was conducted on multistage stochastic resource allocation problems. These experiments were run with multiple commodities, and four different sets of substitution matrices. The substitution matrix gives the fraction of revenue (denoted γ_{kl}) if commodity type k is used to serve task type l (where $\gamma_{kk} = 1$). The approximation dynamic programming formulation was then compared to a rolling horizon procedure that made decisions in each time period by optimizing over the next 20 periods using a (rounded) point forecast of future events. This approach represents standard industry practice for these problems.

Figure 10 shows the comparison for four datasets of the approximate dynamic programming method (using a SAFE algorithm for performing functional approximations) against the rolling horizon procedure. The differences are substantial. These results do assume that if a demand is not served on time then it is lost. Flexibility in the time at which a task can be served would probably reduce this gap.

An experiment was run using the car distribution data from a major railroad. This dataset used actual demand forecasts (and the distributions derived from this forecasting process), an actual supply snapshot and actual transit times. Orders due on one day of the week could be served earlier or later (with a penalty) but could not be carried over to the following week. These runs were done with a system being developed for production use, and are felt to be quite realistic.

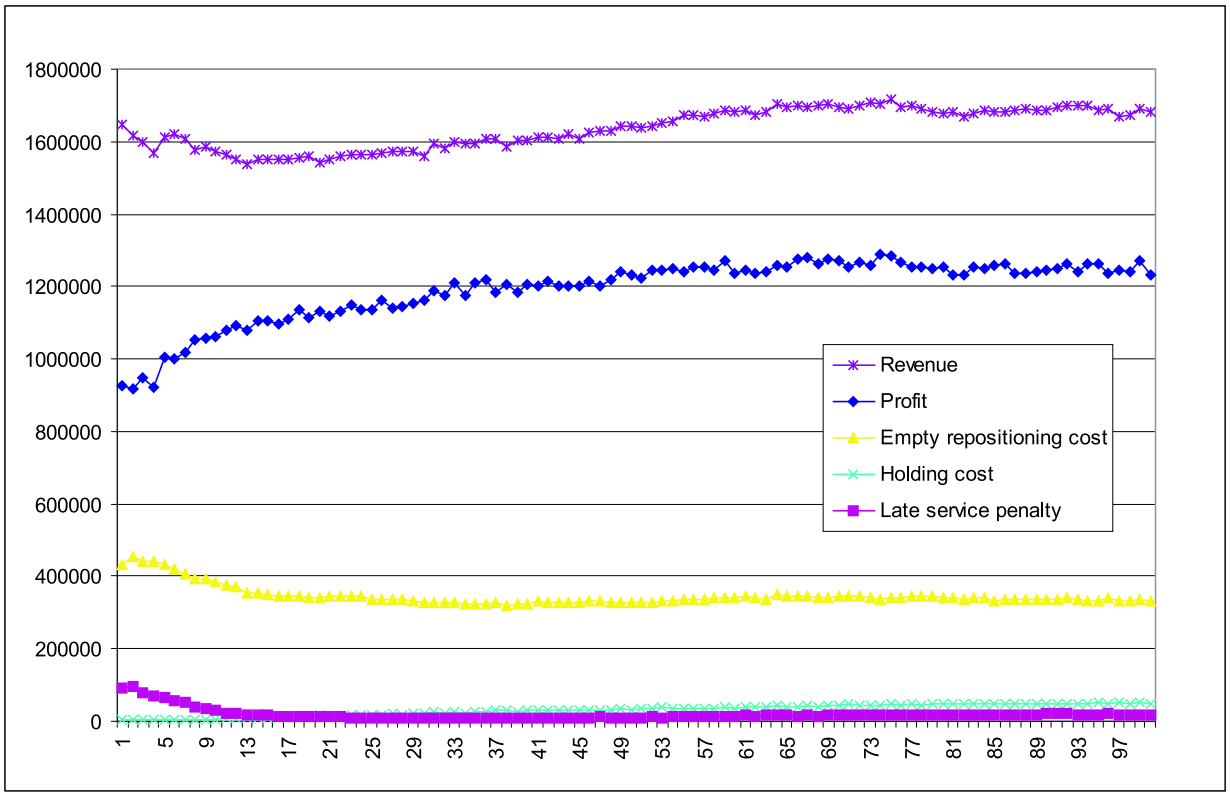

Figure 11: Costs, revenues and profits as the adaptive learning algorithm progresses

The figures that follow show several statistics as the algorithm progresses. The first iteration uses recourse functions equal to zero, and therefore represents a classical myopic model of the sort widely used in industrial practice. Improvements past the first iteration represent a measure of how well our adaptive learning logic performs relative to the standard industrial model.

Figure 11 shows total revenue (the top line), total profits (second line), empty repositioning costs (the third line), service penalties (the fourth line for the first few iterations), and finally the costs of holding cars at certain locations (there are selected locations where there is no holding cost). We note that profits improve steadily over the first fifth iterations before roughly leveling out. Over this range, revenue actually decreases at first, but this is counterbalanced by reductions in empty cost and service penalties. We note that toward the end, the total revenue is slightly higher than it was for the first iteration, indicating that a simple myopic model will do an effective job of covering demand (but just will cover the demands as effectively as the adaptive learning algorithm).

Figure 12 shows empty miles as a percent of total miles (empty miles plus the loaded miles for each car order). This graph indicates a fairly dramatic reduction in empty miles over the first 20 to 30 iterations. Although the total demand served is also dropping in the early iterations, we have expressed empty miles as a percent of the total. It is likely, however, that some of the orders that require longer empty movements (relative to the distance required for the order) are being turned down at first. Since total revenue is slightly higher than

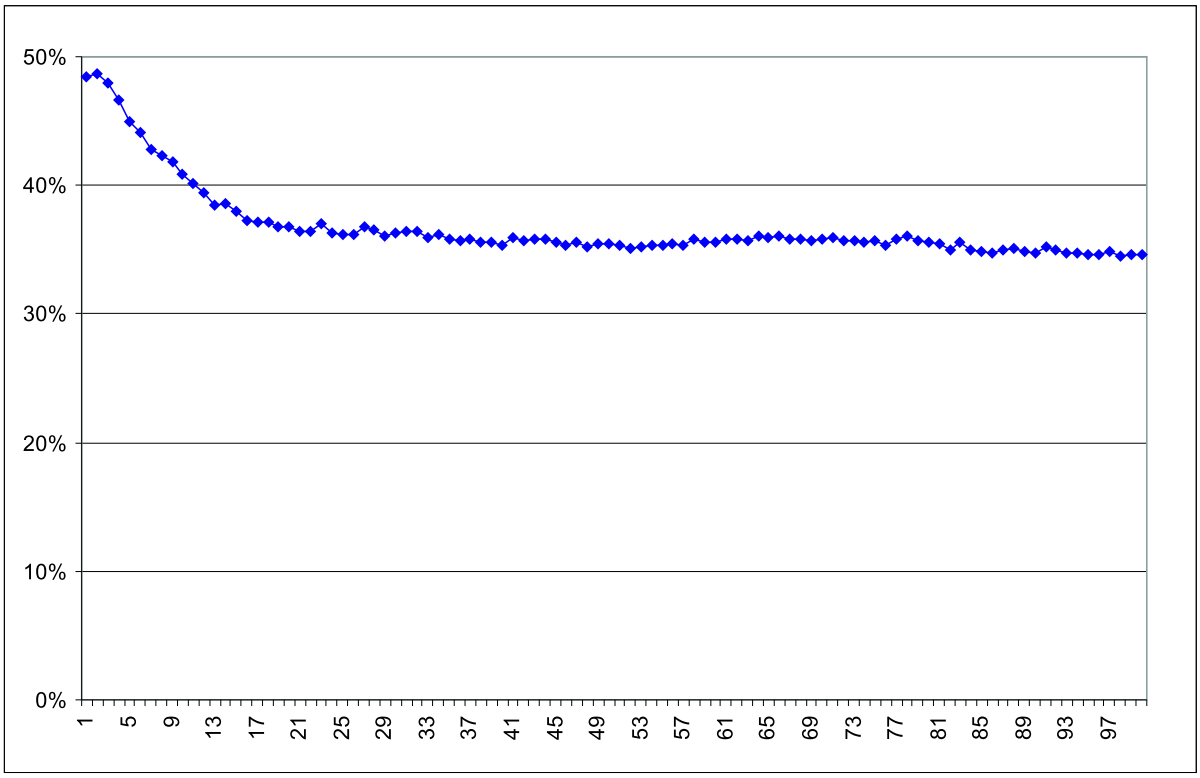

Figure 12: Total empty miles traveled as the learning algorithm progresses

for the first iteration, it appears that the system "learns" how to serve these less profitable orders as the algorithm progresses.

Finally, Figure 13 shows the average days of lateness with which each order is being served. As the algorithm progresses, the lateness drops from an average of almost 3.5 days down to slightly over a day.

7 Implementation issues

Real applications of these methods to actual problems in rail car distribution invariably introduce a set of issues that range from new areas of research to the usual collection of war stories. Some of these include:

1) Storing cars - When there are excess cars, it may be necessary to store cars at predefined locations. One of the challenges of a stochastic model is to store the right number of cars, keeping in mind that we would like to keep enough cars on hand to handle unusual spikes in demand. This is a classic overage/underage problem that stochastic models should excell at. The problem is that the solution depends very much on the cost of overage (what is the cost of keeping idle cars on hand) and the cost of underage (what is the cost of unsatisfied demand). As with most of these problems, the cost

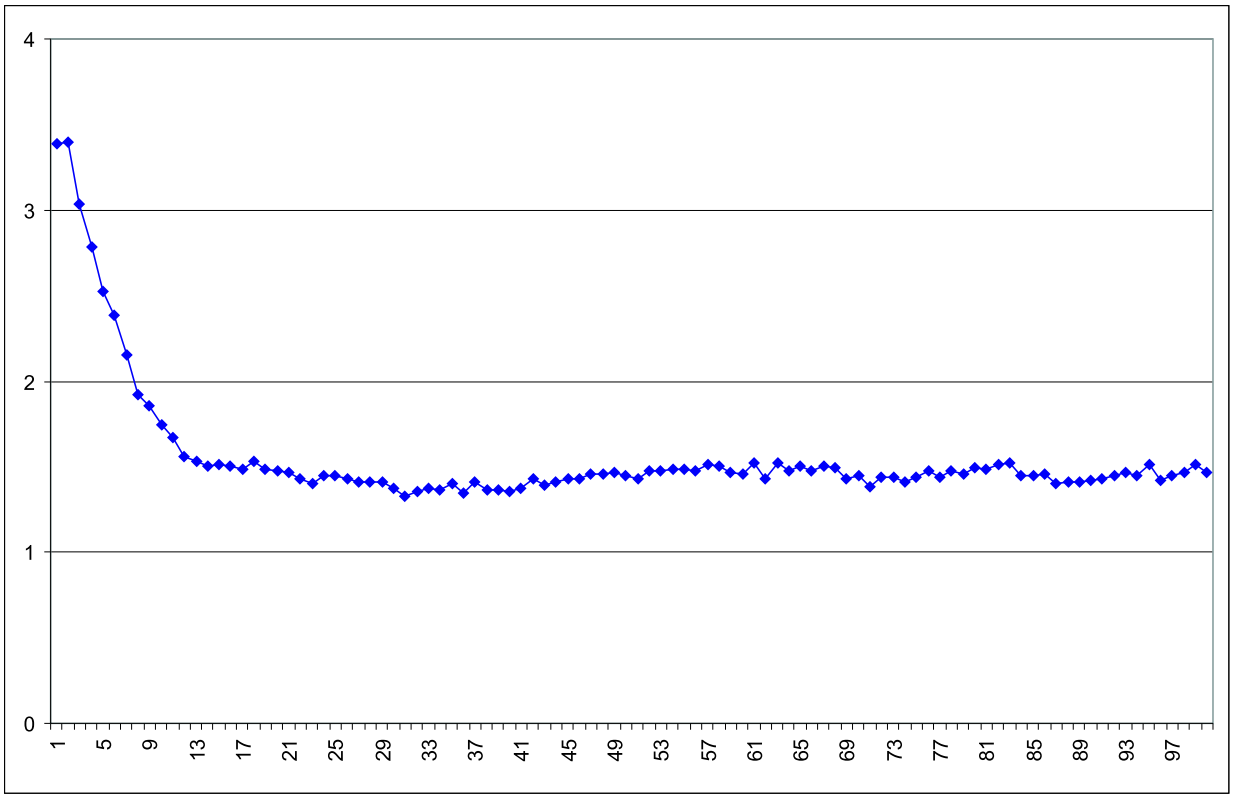

Figure 13: Average days of lateness per order

of unsatisfied demands is difficult to estimate, and often we are choosing numbers to satisfied a target coverage level.

2) Transit times - Transit times are perhaps the Achilles heel of a car distribution system. Stochastic models can, in principle, handle stochastic transit times, but estimating these transit times can be quite a challenge. Standard procedure would be to simply use historical data, but this data can be notoriously unreliable. The solution is to combine engineered transit times (which would normally represent the fastest time a car can achieve between two points), but even this is not entirely reliable. Engineered transit times assume that the car follows a schedule, while in practice some cars will move, even regularly, on special trains that do not appear in the schedule.

3) Repositioning - The heart of any stochastic model involves repositioning cars because of what "might" happen. Planners at railroads do this all the time, but trusting a model to do this is going to be a big step.

4) Train schedules - Our model does not reflect train capacities, which implies that we may try to move cars in a way that results in cars having to wait for available train capacity.

5) Short cuts - The data might say that it takes six days to move a car, while operations will say they can do it in two days by putting the car on a unit train that is not in the

schedule. Although this is a clear data error, it represents one of a host of data issues that will arise in the course of the project.

8 Bibliographic notes

There is a long history of modeling car distribution problems (and related fleet management problems) as deterministic linear programs, initially as pure networks (Feeney (1957), Leddon & Wrathall (1967), Gorenstein et al. (1971), Misra (1972), White (1972), Herren (1973), Herren (1977), White & Bomberault (1969), Mendiratta & Turnquist (1982)) followed by more general models (Haghani (1989)). Recently, Joborn (2001) has developed and implemented a model for car distribution at the Swedish National Railway which considers trains schedules and capacities, producing a relatively large integer multicommodity flow problem. Dejax & Crainic (1987) provide a thorough review of the research in fleet management at the time, covering both rail and intermodal container applications.

Car distribution and similar problems in fleet management were some of the original motivating applications for stochastic programming (Dantzig (1955) and Ermoliev et al. (1976)). Jordan & Turnquist (1983) and Powell (1986) formulated stochastic fleet management problems as nonlinear programming problems using decision variables that sent a fraction of the supply of vehicles through a node to a particular destination. This modeling approach, however, was not able to provide an accurate model of the problem. Crainic et al. (1993) provide a general stochastic, dynamic model for container distribution, but do not provide an algorithm.

Car distribution is a classic multistage stochastic programming problem. Most of the techniques we use are based on results from the theory of stochastic approximation procedures (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)). The car distribution problem represents a class of resource allocation problem that lends itself to a mixture of dynamic programming and stochastic programming formulations.

The detailed study of two-stage stochastic networks was initiated by Wallace (1986), Wallace (1987) and Birge & Wallace (1988) in the context of general two-stage stochastic programs, and independently by Powell (1986), Powell (1987) and Powell (1988), in the context of dynamic fleet management. The latter group of papers eventually led to a line of research (Powell & A. (1998), Powell et al. (2002*b*), Godfrey & Powell (2001), and Godfrey & Powell (2002*a*)) which developed the idea of using Monte Carlo sampling of stochastic gradients to help build up separable, piecewise linear value function approximations. This particular class of functions is particularly convenient for problems requiring integer solutions. Very recent research (Papadaki & Powell (2002), Topaloglu & Powell (2001), Powell et al. (2002*a*)) has begun to formalize the theory behind estimating separable concave approximations, with a growing body of experimental research supporting their effectiveness

(Godfrey & Powell (2002*a*), Godfrey & Powell (2002*b*), Topaloglu & Powell (2000)).

The work in this chapter is based in part on a project to develop and implement a production car distribution system for a major railroad. This project has provided us with a host of real issues involving the modeling of evolving information processes which stochastic programming (broadly defined) can handle. The problem is also attractive because the standard modeling and algorithmic techniques are all based on deterministic approximations.

Acknowledgement

This research was supported in part by grant AFOSR-F49620-93-1-0098 from the Air Force Office of Scientific Research.

References

Bertsekas, D. & Tsitsiklis, J. (1996), *Neuro-Dynamic Programming*, Athena Scientific, Belmont, MA. 31

Birge, J. & Louveaux, F. (1997), *Introduction to Stochastic Programming*, Springer-Verlag, New York. 31

Birge, J. & Wallace, S. W. (1988), 'A separable piecewise linear upper bound for stochastic linear programs', *SIAM J. Control and Optimization* **26**(3), 1–14. 31

Blum, J. (1954), 'Multidimensional stochastic approximation methods', *Annals of Mathematical Statistics* **25**, 737–744. 31

Crainic, T., Gendreau, M. & Dejax, P. (1993), 'Dynamic stochastic models for the allocation of empty containers', *Operations Research* **41**, 102–126. 31

Dantzig, G. (1955), 'Linear programming under uncertainty', *Management Science* **1**, 197–206. 31

Dejax, P. & Crainic, T. (1987), 'A review of empty flows and fleet management models in freight transportation', *Transportation Science* **21**, 227–247. 31

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. 31

Ermoliev, Y. (1988), Stochastic quasigradient methods, *in* Y. Ermoliev & R. Wets, eds, '*Numerical Techniques for Stochastic Optimization*', Springer-Verlag, Berlin. 31

Ermoliev, Y., Krivets, T. & Petukhov, V. (1976), 'Planning of shipping empty seaborne containers', *Cybernetics* **12**, 664. 31

Feeney, G. (1957), Controlling the distribution of empty cars, *in* 'Proc. 10th National Meeting, Operations Research Society of America'. 31

Gladyshev, E. G. (1965), 'On stochastic approximation', *Theory of Prob. and its Appl.* **10**, 275–278. 31

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. 26, 31, 32

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. 32

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. 26, 31

Gorenstein, S., Poley, S. & White, W. (1971), On the scheduling of the railroad freight operations, Technical report 320-2999, ibm philadelphia scientific center, IBM. 31

Haghani, A. (1989), 'Formulation and solution of a combined train routing and makeup, and empty car distribution model', *Transportation Research* **23B**(6), 433–452. 31

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. 31

Herren, H. (1977), 'Computer controlled empty wagon distribution on the SSB', *Rail International* **8**(1), 25–32. 31

Infanger, G. (1994), *Planning under Uncertainty: Solving Large-scale Stochastic Linear Programs*, The Scientific Press Series, Boyd & Fraser, New York. 31

Joborn, M. (2001), Optimization of empty freight car distribution in scheduled railways, Ph.D. thesis, Department of Mathematics, Linkoping University, Sweden. 31

Jordan, W. & Turnquist, M. (1983), 'A stochastic dynamic network model for railroad car distribution', *Transportation Science* **17**, 123–145. 31

Kall, P. & Wallace, S. (1994), *Stochastic Programming*, John Wiley and Sons, New York. 31

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. 31

Mendiratta, V. & Turnquist, M. (1982), 'A model for the management of empty freight cars', *Trans. Res. Rec.* **838**, 50–55. 31

Misra, S. (1972), 'Linear programming of empty wagon disposition', *Rail International* **3**, 151–158. 31

Papadaki, K. & Powell, W. B. (2002), A discrete on-line monotone estimation algorithm, Technical report, Princeton University, Department of Operations Research and Financial Engineering. 31

Powell, W. B. (1986), 'A stochastic model of the dynamic vehicle allocation problem', *Transportation Science* **20**, 117–129. 31

Powell, W. B. (1987), 'An operational planning model for the dynamic vehicle allocation problem with uncertain demands', *Transportation Research* **21B**, 217–232. 31

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. 31

Powell, W. B. & A., C. T. (1998), 'Dynamic control of logistics queueing network for large-scale fleet management', *Transportation Science* **32**(2), 90–109. 31

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. 31

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. 31

Puterman, M. L. (1994), *Markov Decision Processes*, John Wiley and Sons, Inc., New York. 31

Robbins, H. & Monro, S. (1951), 'A stochastic approximation method', *Annals of Math. Stat.* **22**, 400–407. 31

Sutton, R. & Barto, A. (1998), *Reinforcement Learning*, The MIT Press, Cambridge, Massachusetts. 31

Topaloglu, H. & Powell, W. B. (2000), Dynamic programming approximations for stochastic, time-staged integer multicommodity flow problems, Technical Report CL-00-02, Department of Operations Research and Financial Engineering, Princeton University. 27, 32

Topaloglu, H. & Powell, W. B. (2001), An algorithm for approximating piecewise linear functions from sample gradients, Technical report, Princeton University, Department of Operations Research and Financial Engineering. 31

Wallace, S. (1986), 'Solving stochastic programs with network recourse', *Networks* **16**, 295–317. 31

Wallace, S. (1987), 'A piecewise linear upper bound on the network recourse function', *Mathematical Programming* **38**, 133–146. 31

White, W. (1972), 'Dynamic transshipment networks: An algorithm and its application to the distribution of empty containers', *Networks* **2**(3), 211–236. 31

White, W. & Bomberault, A. (1969), 'A network algorithm for empty freight car allocation', *IBM Systems Journal* **8**(2), 147–171. 31