

Dynamic Models for Freight Transportation

Warren B. Powell Belgacem Bouzaiene-Ayari
Hugo P. Simao
Department of Operations Research and Financial Engineering
Princeton University

September 14, 2005

Abstract

Freight transportation is characterized by a variety of dynamic information processes: the demands of customers, the performance of people and equipment, and the performance of the network. In addition, large networks are characterized by how they organize decisions and information: who makes what decision with what information? This chapter approaches the development of models for freight transportation, focusing on the theme of capturing the organization and flow of information and decisions. Decisions can be made with up to four classes of information, providing a spectrum of possibilities when designing decision functions which capture the information that is actually available to decision makers.

1 Introduction

Dynamic models arise in a vast array of transportation applications as a result of the need to capture the evolution of activities over time. But exactly what do we mean by a “dynamic model”? All dynamic models capture the timing of physical activities which introduces both modeling and algorithmic challenges and opportunities. In real problems, there are three classes of activities that can evolve over time: the physical movement of freight, equipment and people; the evolution of information; and the movement of financial resources as services are paid for.

In this chapter we focus on modeling the organization and flow of information and decisions, in the context of freight transportation problems that involve the management of people and equipment to serve the needs of customers. The timing of the flow of capital is becoming an increasingly important dimension of freight transportation, but as of this writing, there has been virtually no formal research on the topic. Modeling the timing of physical activities, by contrast, dates to the 1950’s. These models introduce a range of modeling and algorithmic challenges that have been studied since the early years of operations research models.

Our decision to focus on modeling the evolution of information (or more broadly, the organization and flow of information and decisions) reflects the growing maturity of this class of models, and the importance of questions that require an explicit model of information processes. Often categorized under the phrase “stochastic, dynamic models,” there are two issues that arise in this context:

- Improving the management of physical resources by using a more accurate representation of information processes.
- Designing and controlling information processes to maximize performance.

The first class of questions focuses on more realistic models to improve the design and control of physical systems. Examples of questions include:

- How do we best allocate rail cars to handle future requests?
- Which driver should move a load of freight given the as-yet unknown loads that will be available at the destination of the first load?

- How many dock workers should be assigned to work a shift given the uncertain amount of freight to be unloaded?
- How many locomotives should be held at a yard to handle potential equipment failures of locomotives that are expected to arrive inbound over the next 12 hours?

All of these are examples of questions concerning the allocation of physical resources in the presence of uncertainty in the future. Most often, the uncertainty arises around demands that are placed on the system, but there can be uncertainty in the availability of resources (how many drivers can the trucking company hire this week? will the locomotive break down?) and the performance of the network itself (primarily travel times). A common theme when designing and controlling systems under uncertainty is producing solutions that work well over as broad a range as possible of potential outcomes in the future. Such solutions are referred to as *robust*, although this term is most often used in the context of design problems.

The second class of questions arises in the design and control of information processes. Examples include:

- Should a company invest in wireless communication devices that allow them to determine the location and status of drivers and equipment?
- Should a company invest in a database that tracks the status of all of its drivers/crew?
- What is the cost of allowing customers to make requests at the last minute?

Questions such as these require a model that allows us to explicitly manipulate the information process. If we consider a scenario which reduces the information available when we are making a decision, then we have to be sure that we are producing the best decisions that we can even if we do not have the information.

2 Some illustrative applications

It helps to begin our presentation with a discussion of different applications in freight transportation. Our presentation of models focuses on the resources being managed, and the information and decision processes that govern the evolution of our systems. Below we list a series of application

areas, along with examples of resource classes, (exogenous) information classes, and decision classes. For each subcategory, we briefly list typical examples of each class. For transportation problems, every resource includes as attributes its location and the estimated time at which the resource will arrive at the destination (later, we provide a more formal and general characterization of “estimated time of arrival”). For this reason, we exclude these attributes in the discussion below.

Truckload trucking

Resource classes

- Drivers, who are normally modeled as including the tractor. Drivers are characterized by attributes such as location, home domicile, time due at home, international experience, and sleeper vs. single driver.
- Trailers: loaded status (loaded or empty), cleanliness, repair status.
- Loads of freight: destination, service requirements, equipment needs, shipper classification.

Information classes

- Customer requests for service.
- Drivers entering and leaving the system.
- Equipment status.
- Transit times.

Decision classes

- Drivers: couple with trailer and/or load, move empty, move loaded, go on rest (at home or away).
- Trailers: move (empty or loaded), clean, repair.
- Loads: accept/reject initial request, spot pricing of a load, couple with trailer, move (with trailer), move using a subcontractor (a driver not managed by the company).

Rail car distribution

Resource classes

- Cars: car type, loaded status, maintenance status, cleanliness, shipper pool, owner.
- Orders: destination, service requirements, commodity type and substitutability.

Information classes

- Customer orders, including updates to customer orders.
- Empty cars becoming available from shippers or other railroads.
- Transit times.
- The destination of an order (revealed only after a car is filled).
- Acceptability of a car to the customer (is it clean? in good repair?).

Decision classes

- Cars: Move (loaded or empty), repair, clean, switch pools, move off-line to another railroad, spot pricing of an order.
- Orders: accept/reject, subcontract.

Locomotive management

Resource classes

- Locomotives: type, horsepower, high/low adhesion, fuel status, due-in-shop date, inbound train that the locomotive last pulled (determines the other locomotives the unit is connected to).
- Trains: train type (grain, coal, merchandise, stack, priority), destination, departure time, tonnage, horsepower required per ton being pulled.

Information classes

- Locomotives: maintenance status, arrivals to system, departures from system.
- Trains: tonnage, additions to schedule, cancellations (annullments), delays.

Decision classes

- Locomotives: assignments to trains, assignment to shop, repositioning (moving without a train).
- Trains: decisions to delay trains.

Less-than-truckload trucking

Resource classes

- Shipments: destination, service requirement, size (weight and density).
- Trailers: type (typically 28' or 48'), current load (shipments on the trailer), destination, departure time.

- Drivers: domicile, bid vs. nonbid, sleeper vs. single, days away from home, hours of service.
- Tractors: type, status.

Information classes

- Shipments entering the system.
- Changes in driver status.
- Transfer of shipments at sort facilities.

Decision classes

- Where to run trailers direct.
- What trailer to put a shipment on.
- Coupling of trailers into loads to be moved (typically involves pairing 28' trailers).
- Assignment of drivers to loads.

Intermodal container operations

Resource classes

- Containers: type, loaded status, cleanliness, repair status, ownership.
- Loads: destination, weight, service requirements, routing restrictions.

Information classes

- Customer orders.
- Transit times.
- Movement of containers.

Decision classes

- Customer orders: accept/reject, move via subcontractor, spot pricing of an order.
- Containers: assignment to order, movement (loaded or empty), assignment to ship or train departures, intra-port management, routing and scheduling by truck.

Air cargo

Resource classes

- Aircraft: type, configuration, load status, maintenance schedule.

- Requirements (loads of freight): destination, weight, size characteristics, passengers, pickup and delivery windows.
- Pilots: training, home location, hours worked, work schedule.

Information classes

- Requests to move freight and passengers.
- Equipment failures (and random changes in status).
- Weather delays.

Decision classes

- Move aircraft from one location to another.
- Reconfigure aircraft (to handle passengers or different types of cargo).
- Repair, refuel.
- Outsource customer requests.

3 A resource model

At the heart of our problems is the challenge of modeling resources. Although the focus of this chapter is information, we are interested in information in the context of managing resources for freight applications.

We begin in section 3.1 by addressing the simple question: what is a resource? Section 3.2 introduces a basic resource model which gives us a general notational framework for modeling resource classes and attributes. Section 3.3 introduces the concept of resource layers, which allows us to handle combinations such as driver/trailer/load and locomotive/train; these are composite resources that take on new behaviors. Section 3.4 notes the interaction between the design of the attribute space, and the flow of information and decisions. Section 3.5 shows how our attribute notation can be used to describe six classes of resource management problems. Finally, section 3.6 closes with a discussion of three perspectives of the “state” of our system.

3.1 What is a resource?

Any freight transportation system can be thought of as a problem in the management of resources. When we put it this way, we tend to think of resources as people and equipment, and sometimes

fixed facilities. When we formulate optimization models, resources tend to appear as right-hand side constraints, which suggests that resources are anything that constrains the system. From the perspective of models, we can divide all data into information classes, where “resources” are information classes that constrain the system. It is useful to distinguish between static resource classes (such as terminals) which constrain the system, but which may not themselves be decision variables, and active resource classes (people, equipment) which we are managing.

When we adopt this more formal definition of a resource, we find that “demands” (also known as customers, tasks, requirements, loads, moves, pickups/deliveries, schedules) are also a form of “resource.” Often, the distinction between “managing resources” (objects over which we have control) to “serve customers” (objects over which we have no control) is an artificial one that does not stand up to scrutiny (and certainly not in the formalism of a mathematical model).

A simple example illustrates why it is useful to use a broader definition of resources. Consider the problem faced by a truckload carrier of moving a load from city i to city j . The driver at i needs to return to his home domicile in city d . For this reason, the carrier assigns the driver to the load but moves both to an intermediate relay r , where the driver and load separate. The driver is then assigned to another load that moves him closer to his home, while the load is assigned to a new driver who will presumably move it toward its final destination (in industries such as long-haul less-than-truckload trucking, a load can move through four or five such relays before reaching its destination). During this time, we are trying to get the driver to his home and the load to its destination. While the attributes of each are different, the modeling issues are the same.

A working definition of a resource is *an information class that constrains the system*. This broad definition includes not only equipment and people, but also fixed assets such as truck terminals and railroad track which limits our ability to move freight. An *active* resource class is *an endogenously controllable information class that constrains the system*. Drivers, trailers, locomotives and planes are all examples of active resource classes. A load of freight also limits our system (it limits how many trucks we can move loaded). In most models, a load is an example of a *passive* resource class since it constrains the system, but we cannot manage it directly (we can move a trailer with a load in it, but not a load without a trailer). Exceptions arise with companies that can contract out movements. In this case, we can effectively move a load without using any of the company-owned drivers or equipment. In such a setting, the load becomes an active resource class.

3.2 A basic resource model

Our resource model, then, begins with a listing of the resource classes:

\mathcal{C}^R = Set of resource classes.

We describe a resource using a vector of attributes:

a = Vector of attributes which describe the state of a resource.

\mathcal{A}^c = Space of possible attribute vectors of a resource in class $c \in \mathcal{C}^R$.

Attributes can be categorical (home domicile, equipment type, maintenance status) or numerical (hours worked, days until next maintenance stop, fuel level). Attribute vectors can be divided between static attributes (for example, the type of equipment or domicile of a driver) and dynamic attributes (location, maintenance status, hours of service). Often, the set of static attributes is combined to identify a resource type or commodity, while the set of dynamic attributes is the state. *Dynamic* resources have at least one dynamic attribute; *static* resources are characterized by attributes that are completely static.

We describe the status of all the resources in our system using:

R_{ta}^c = The number of resources in class $c \in \mathcal{C}^R$ with attribute a at time t .

$R_t^c = (R_{ta}^c)_{a \in \mathcal{A}^c, c \in \mathcal{C}^R}$.

Recall that one of our resource classes is the customer demands. It is common to model customer demands as the only source of exogenous information (new requests being made on the system) but there are many problems where there are exogenous updates to the other resources in the system (drivers being hired, equipment breaking down, equipment arriving exogenously to the system). We capture all these forms of information using

\hat{R}_{ta} = Change in R_{ta} between $t - 1$ and t due to exogenous information.

$\hat{R}_t = (\hat{R}_{ta})_{a \in \mathcal{A}}$.

3.3 Resource layering

It is often the case that resources from different classes will be *coupled* to form a *layered resource*.

For example:

- When managing locomotives, a set of locomotives will be assigned to a train to move cars from one location to the next. When a locomotive is attached to a train, it can only be removed at a cost, so the behavior of a locomotive at a location depends on whether it is attached to a train or not. Furthermore, the ability to route a locomotive back to its home shop is very much affected by the attributes of a train that it is attached to (a railroad might easily decide not to detach a locomotive from a train that it is pulling, even if the destination of the train takes the locomotive away from its maintenance base).
- A cargo aircraft loaded with freight that breaks down will have a different behavior from an empty aircraft.
- Decisions about a business jet, with a particular pilot moving a passenger, that has just broken down will depend on the attributes of the aircraft, pilot and passenger.

We have to define the ways that resources may be coupled, which produces a set of resource layers. A layer, in general, is a resource class comprised of the coupling of one or more resource classes. We might have two resource classes (pilots and aircraft) and two resource layers. The first layer might be just the pilots (P) while the second might be aircraft with pilots ($A|P$).

To clearly distinguish between the resource and the resource layer, we use the same letter as a resource surrounded by a pair of parentheses to denote that resource layer. We let

$(A) = A|P$ denotes the aircraft layer.

$(P) = P$ denotes the pilot layer.

$\mathcal{L} = A$ layering, representing the set of resource layers.

$= ((A), (P)).$

$= \{A|P, P\}$

$l =$ The index for resource layer, $l \in \mathcal{L}$, i.e. $l = A|P$ or $l = P$

For this example, we would refer to the pilot layer as a primitive layer, while the aircraft layer is a composite layer. The attributes of each layer are given by

$$\mathcal{A}^{(P)} = \mathcal{A}^P, \text{ the attribute space of a pilot layer.}$$

$$a^{(P)} = a^P, \text{ the attribute vector of a pilot layer.}$$

$$\mathcal{A}^{(A)} = \mathcal{A}^{A|P}.$$

$$= \mathcal{A}^A \times \mathcal{A}^P, \text{ the attribute space of an aircraft layer.}$$

$$a^{(A)} = a^{A|P}.$$

$$= a^A | a^P \in \mathcal{A}^{(A)}, \text{ the attribute vector of an aircraft layer.}$$

It is very common in transportation and logistics to model one or two-layer problems. A good example of a one-layer problem is the assignment of aircraft to a fixed schedule of flights (where the departure time of the flight is fixed). A two-layer problem would arise when assigning truck drivers to loads of freight, where we have to determine when the load should be moved (some loads have 24 hour windows or longer). This would be an example of a two-layer problem with a single active layer (the driver) and a single passive layer (the load). In many companies, it is possible to outsource a load, which is equivalent to moving a load without a driver, giving us an instance of a two-layer problem with two active layers. A three-layer problem might arise when we have to simultaneously manage drivers, tractors and trailers, or pilots, aircraft and loads of cargo.

Published examples of three-layer problems are fairly rare, but a good example is the NRMO model for the military airlift problem (Morton et al. (1996), Baker et al. (2002)) which tracks the flows of aircraft, crews and cargo in a military airlift problem.

3.4 Attributes, decisions and information

The attribute space \mathcal{A} should not reflect every possible state that a resource may occupy, but rather the states where the arrival of new information, and/or the ability to make a decision (which, after all, represents the arrival of new information) might affect the behavior of the resource. For example, consider a military aircraft which has to move from the United States to India through a series of three intermediate airbases (for refueling and potential repairs). We would need to represent the attributes of the aircraft at any point where new information might change the movement of the aircraft. For example, equipment breakdowns, congestion at the airbase or weather delays can

affect the timing of the route. Furthermore, this new information might result in decisions being made to reroute the aircraft. In the simulation package AMOS, used by the air mobility command, it is common not to model new information or decisions while an aircraft is enroute, in which case we only would need to model the aircraft at the end of the route.

The role of information and decisions in the design of the attribute space is especially important in the presence of layered resources. It is clear the attribute space of a layered resource is much more complex than that of either primitive resource. The question is: what do we need to model? Consider a simplified model of a fleet management problem where a vehicle (truck, jet, container) is assigned to move a request from one location to another. There are no intermediate stops, and no decisions can be made about the equipment being managed until it has finished a task. We might refer to this as a “two-layer problem” but we would never need to actually model a layered resource. We do not make any decisions about a piece of equipment until it has finished a task. While there is a period of time when the equipment is coupled with the customer request, there is no need to explicitly model this.

3.5 Major problem classes

We can use our attribute vector to describe six major problem classes. These are:

- **Inventory problems** - $a = \{\}$. This basic problem arises in classical inventory theory, where there is a single type of product being held in inventory. The attribute a is a null vector.
- **Multiproduct inventory** - $a = \{k\}$ where $k \in \mathcal{K}$ is a set of resource types (product classes). The attribute a consists of a single, static element.
- **Single commodity flow problems** - $a = \{i\}$ where $i \in \mathcal{I}$ is a set of states or locations of a piece of equipment. An example is managing a fleet of identical trucks, where the only attribute of a truck is its current location.
- **Multicommodity flow** - $a = \{k, i\}$ where $k \in \mathcal{K}$ represents types of resources and $i \in \mathcal{I}$ is a set of locations or states.
- **Heterogeneous resource allocation problems** - $a = (a_1, \dots, a_n)$. Here we have an n -dimensional attribute vector, where a_i is the i^{th} attribute of a . These applications arise in the management of people and complex equipment.

- **Multilayered resource scheduling problems** - $a = \{a^{c_1}|a^{c_2}|\dots|a^{c_n}\}$. Now the attribute vector is a concatenation of attribute vectors.

These six problem classes represent an increasing progression in the size of the attribute space. The problems with smaller attribute spaces are generally viewed as flow problems, even if we require integer solutions.

3.6 The states of our system

In dynamic models, it is common to talk of the “state of the system,” but it is very hard to define a state formally. A review of virtually any of the major textbooks on dynamic programming will fail to produce a formal definition of the state of the system.

We find that there are three perspectives of state variables. The first is the attribute vector a which can be thought of as the state of a single resource. In fields such as crew scheduling which make use of column generation methods (Desrosiers et al. (1995), Vance et al. (1997), Desrochers & Soumis (1989)), the attribute vector is referred to as a label, and the determination of the best schedule for a crew is formulated as a dynamic program over the state space \mathcal{A} .

The second is the vector R_t which represents the resource state of the system (in most dynamic programming applications, R_t is viewed as the state variable). While the size of \mathcal{A} can range from one to the millions (or larger), the number of possible values of R_t can be truly huge. Let \mathcal{R} be the space of possible values of R_t (which we assume is discrete), and let $N = \sum_{a \in \mathcal{A}} R_{ta}$ be the total number of resources being managed. Then it is straightforward to show that

$$|\mathcal{R}| = \binom{N + |\mathcal{A}| - 1}{|\mathcal{A}| - 1}. \tag{1}$$

Even for small problems, this number can be extremely large (creating problems where it exceeds 10^{50} is not that difficult). For practical purposes, it is useful to refer to such spaces as “effectively infinite.”

The third state variable might be referred to as the “information state,” although a more precise definition is the “knowledge state.” Consider a problem where there is a series of customer demands \hat{D}_t which vary from period to period. We might estimate the mean demand using a simple

exponential smoothing model

$$\bar{\mu}_t = (1 - \alpha)\bar{\mu}_{t-1} + \alpha\hat{D}_t.$$

In addition, we might estimate the variance using

$$\bar{\sigma}_t^2 = (1 - \alpha)\bar{\sigma}_{t-1}^2 + \alpha(\hat{D}_t - \bar{\mu}_{t-1})^2.$$

At time t , $(\bar{\mu}_t, \bar{\sigma}_t^2)$ would be part of what we know at time t . Given $(\bar{\mu}_t, \bar{\sigma}_t^2)$, we have all the information we need from history about customer demands. Our state variable would then be represented by $S_t = (R_t, \bar{\mu}_t, \bar{\sigma}_t^2)$.

3.7 Bibliographic notes

The mathematical properties of pure networks were discovered in the 1950's (see Ford & Fulkerson (1962) for a classic early description of networks), but it was in the 1970's that Glover, Klingman and Kennington discovered and exploited the intersection of network algorithms and emerging computer science techniques to produce exceptionally fast (even in those days) algorithms for solving large networks (see Glover et al. (1974), Langley et al. (1974); Glover et al. (1992) summarizes this body of work). While there were specialized problems that fit this model in transportation (White (1972)) most problems in practice required tracking different types of equipment, where there were substitution opportunities between equipment types. The numerous applications of these *multi-commodity flow problems* produced a vast literature which sought to exploit the embedded network structure to overcome the otherwise large size of these problems (see, for example, Lasdon (1970), Kennington (1978), Kennington & Helgason (1980), Assad (1978)). A significant development in the multicommodity literature was its adaptation to solving routing and scheduling problems for vehicles and crews, where the latter are typically characterized by complex vectors of attributes (Desrosiers et al. (1984), Lavoie et al. (1988), Desrochers & Soumis (1989), Barnhart et al. (1998), Vance et al. (1997)). These techniques involve solving a multiattribute dynamic program for a single crew; a master program then chooses the best from a set of potential schedules to produce an optimal overall schedule. The attribute vector appears only in the dynamic program solved in the subproblem; once the potential tours are generated, the master problem is typically a set partitioning or set covering problem which is a very special type of integer multicommodity flow problem. Powell et al. (2002) formally propose the "heterogeneous resource allocation problem"

which is a generalization of the classical multicommodity flow problem. The modeling framework in this paper is based on Powell et al. (2001) which introduces a general problem class termed the “dynamic resource transformation problem.”

4 Modeling exogenous information processes

The modeling of the flow of information can be surprisingly subtle. In a deterministic model, we might try to solve a problem that looks like

$$\min_x \sum_{t \in \mathcal{T}} c_t x_t$$

subject to

$$\begin{aligned} A_t x_t - B_{t-1} x_{t-1} &= \hat{R}_t \\ x_t &\geq 0 \end{aligned} \tag{2}$$

In a model such as this, we would understand that $x_t = (x_{tij})_{i,j \in \mathcal{I}}$, where x_{tij} is the flow from city i to city j starting at time t (and arriving at j after some travel time τ_{ij}). In other words, x_t is the vector of activities *happening* at time t . We would assume that c_t are the costs incurred in that time period. In other words, in a model such as this, the index t refers to when things are *happening*.

In a model that reflects dynamic information processes, we have to combine models of physical activities over time with models of information arriving over time. Consider a problem that arises in freight car distribution. If we were to look at an order in a historical file, we would simply see a movement from one location to another at a point in time. But if we were to capture the evolution of information that occurred before this physical event, we would see a series of information events as depicted in figure 1, which reflects: 1) the initial call that there is an order requiring a car; 2) the amount of time required to move the empty car to the order; 3) the acceptance of the car by the shipper (who will sometimes reject a car because it is damaged or dirty); 4) the time required to fill the car, and 5) the destination of an order (which is not revealed until the car is loaded).

In our model above, the right hand side constraint \hat{R}_t in equation (2) would normally be interpreted as the new arrivals to the system at time t . However, we need to distinguish between

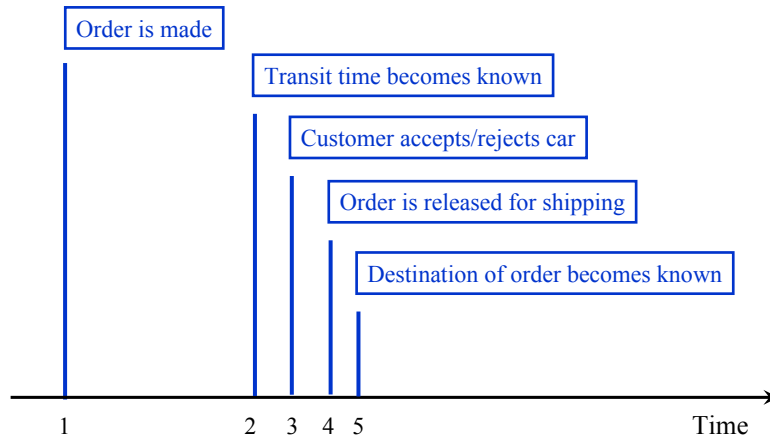


Figure 1: The evolution of information for a single order in a freight car application

the “phone call” representing the information about a new order and the physical arrival of a new order.

In the remainder of this section, we set up the fundamentals for modeling dynamic information processes. Section 4.1 begins by establishing our convention for modeling time. Then, section 4.2 introduces a notational style for modeling information processes. Section 4.3 introduces the important concept of lagged information processes that arises in almost all transportation problems. Finally, section 4.4 provides an introduction to an important mathematical concept used widely in the probability community for modeling information.

4.1 Modeling time

For computational reasons, it is common to model problems in transportation and logistics in discrete time, but it is apparent that in real problems, activities happen continuously. Surprisingly, the modeling of time (which is fairly obvious for deterministic problems) is not standard in the stochastic optimization community (for example, some authors like to index the first time period as $t = 0$ while others choose $t = 1$).

For computational reasons, we model our decision making process in discrete time (these can be fixed time periods such as hourly or daily, or driven by information events). However, it is useful to think of information arriving in continuous time, which resolves any ambiguity about what information is available when we make a decision. Figure 2 shows the relationship between

discrete and continuous time which is most commonly used in the stochastic process community (Cinlar (2003)). In this convention, time $t = 0$ refers to “here and now,” while discrete time period t refers to the time interval $(t - 1, t]$.

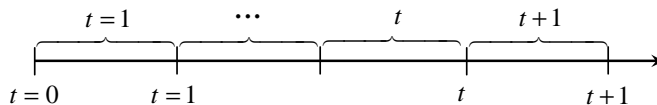


Figure 2: The relationship between continuous and discrete time

It is very important to identify the information content of a variable. In a purely discrete time model, the precise sequencing of decisions and information is ambiguous (was a decision at time t made before or after the information that arrived at time t ?). For this reason, we will view information as arriving continuously over time, while a decision will be made at a discrete point in time. Since a state variable is the information we need to make a decision, it will also be measured in discrete time.

4.2 The information process

In the math programming community, it is common to use x_t as a generic decision variable (x_t might come in different flavors, or we might include y_t and z_t), but there is not a commonly accepted “generic variable” for information. For our presentation, we adopt the convention that W_t is a variable representing all the information that arrived during the time interval between $t - 1$ and t . This information can be about new customer orders, transit times, costs, and the status of people and equipment. We may in general define:

\mathcal{C}^I = The set of dynamic, exogenous information classes, representing different types of information arriving over time (prices, demands, equipment failures, travel times).

W_t^c = The information arriving for class c .

$W_t = (W_t^c)_{c \in \mathcal{C}^I}$.

In a particular application, it will be common to model specific information processes and give each of these names such as demands or prices. As a result, it may not be necessary to use the generic “ W_t ” notation. But it can be useful to have a generic information variable (as it will be in our

presentation) since it will allow us to add new information processes without changing our basic model. We adopt the notation that with the exception of our generic information variable W_t , we use hats to indicate exogenous information processes. For example, if a trucking company faces random demands and spot prices, we might let \hat{D}_t and \hat{p}_t be the demands and prices that we learn about during time interval t (between $t - 1$ and t). Thus we would write $W_t = (\hat{D}_t, \hat{p}_t)$. Given our convention for labeling time, W_1 is our first set of information arriving (W_0 has no meaning since information arrives over time intervals, and the first time interval is $t = 1$).

We further adopt the notation that any variable with subscript t uses information up through time t . This notation is of fundamental importance, and represents a significant departure from standard deterministic dynamic models where t represents when a physical activity is happening. Thus, x_t is a decision that uses information up through time t ; S_t might be a state variable at time t (which uses W_t), and $C_t(x_t)$ is the cost computed using information up through time t .

Normally, models that capture dynamic information processes do so because we do not know what information is going to arrive in the future. This is fundamental to stochastic models, but we wish to emphasize that there are numerous engineering applications which explicitly model dynamic information, but model only one instance of information (for example, what might have occurred in history). When we wish to model multiple information processes, we can adopt the standard notation of the probability community. Let ω represent a particular realization of the information process: $W_1, W_2, \dots, W_t, \dots$, and let Ω be the set of all possible realizations (we now see the motivation for using W_t as our generic variable for information - it “looks like” ω).

It is typically the case that we will be working with a single realization at a time. We can think of ω as an indexing of all the possible outcomes (as in the integers from 1 to $|\Omega|$), or as the literal outcomes of all the possible pieces of information that become known. When we are using Monte Carlo sampling techniques, we might think of ω as the random number seed to generate all the possible random variables. These are all effectively equivalent views, but we suggest that it is generally best to think of ω as a simple index. When we want to refer to the information that becomes known during time interval t , we may write $\omega_t = W_t(\omega)$. In this case, ω_t is a sample realization of all the different sources of exogenous information. If we view ω as the actual information in a realization, we can write

$$\omega = (\omega_1, \omega_2, \dots, \omega_t, \dots).$$

Some authors use ω or ω_t as if they are random variables, while the probability community insists that ω is strictly a sample realization. Readers new to the field need to understand that there is an astonishing lack of unanimity in notational styles. For example, the probability community will insist that an expression such as $Ef(x, \omega)$ has no meaning (ω is not random, so the expectation is not doing anything), while others have no objection to an expression such as this. The debate can be avoided by writing $Ef(x, W)$ where W is our random “information variable.” Readers in engineering might feel that such debates are somewhat pedantic, but the issues become more relevant for researchers who wish to make fundamental theoretical contributions. Our position is to adopt the simplest possible notation that is as mathematically correct as possible.

In time-staged problems, it is often necessary to represent the information that we know up through time t . For this, we suggest the notation:

$$\begin{aligned} H_t &= \text{The history of the process, consisting of all the information known through time } t. \\ &= (W_1, W_2, \dots, W_t). \end{aligned}$$

$$\begin{aligned} \mathcal{H}_t &= \text{The set of all possible histories through time } t. \\ &= \{H_t(\omega) | \omega \in \Omega\}. \end{aligned}$$

$$\begin{aligned} h_t &= \text{A sample realization of a history,} \\ &= H_t(\omega) \end{aligned}$$

Later, we are going to need to define the subset of Ω that corresponds to a particular history. For this purpose, we define

$$\Omega_t(h_t) = \{\omega | (W_1(\omega), W_2(\omega), \dots, W_t(\omega)) = h_t, \omega \in \Omega\}. \quad (3)$$

The notation for modeling the information up to a particular time period is not standard, and readers need to be prepared for a range of different notations in the literature on stochastic optimization.

Our interest is in systems where exogenous information and decisions occur sequentially. It might be useful for some readers to think of “decisions” as *endogenously controllable information classes*. If ω_t is a sample realization of the information arriving during time interval t , then x_t is the decision made at time t (using the information up through time t). Let S_t be the state of

our system immediately before a decision is made (which means that it also contains ω_t). It is customary to represent the evolution of the state variable using

$$S_{t+1} = S^M(S_t, x_t, \omega_{t+1}). \quad (4)$$

The superscript “ M ” refers to “model” since many authors refer to equation (4) as the “plant model” or “system model.” We note that some authors will write equation (4) as $S_{t+1} = S^M(S_t, x_t, \omega_t)$. This means that t refers to when the information is used, rather than the information content (we refer to this as an *actionable* representation, while equation (4) is an *informational* representation). For simple problems, either representation works fine, but as richer problems are addressed, we believe that the informational representation will prove more effective.

The state variable S_t is the information available right before we make a decision, so we might call it the pre-decision state variable. For algorithmic purposes, we sometimes need to use the post-decision state variable (denoted S_t^x), which is the state right after we make a decision, before any new information arrives. The relationship between pre- and post-decision state variables is described using

$$S_t^x = S^{M,x}(S_t, x_t) \quad (5)$$

$$S_{t+1} = S^{M,W}(S_t^x, W_{t+1}). \quad (6)$$

This representation becomes particularly useful when using approximate dynamic programming methods (see section 8.3). Finally, it is sometimes convenient to work only with the post-decision state variable. If this choice is made, we suggest simply defining S_t to be the post-decision state variable (without the superscript x), and writing the transition function using

$$S_t = S^M(S_{t-1}, \omega_t, x_t).$$

Note that we are using the same function $S^M(\cdot)$, but we change the order of the information and decision variables, reflecting the order in which they occur.

4.3 Lagged information processes

When we are solving deterministic models, we assume all information is known in advance. By contrast, most stochastic models assume that the information about an event (in particular a

customer demand) becomes known at the same time as the event actually happens. It is often the case, however, that the information about an event arrives at a different time than the physical event. The most common example is customers calling in orders in advance, but there are examples where the information arrives after an event (the dispatch of a truck might be entered into the computer hours after a truck has left).

We refer to these problems as instances of lagged information processes, reflecting the more common instance when information arrives before an event. This is quite common in freight transportation, and represents an important area of study. For example, carriers may be able to charge extra for last-minute requests, so they need to be able to estimate the cost of the higher level of uncertainty.

In general, a customer might call in a request for service, and then further modify (and possibly cancel) the request before it is finally served. For our purposes, we are going to treat the information in the initial phone call as accurate, so that when we receive information about a future event we can then model that event as being fully known. We introduce two terms that help us with these problems.

Knowable time - This is the time at which the information about a physical resource arrives.

Actionable time - This is the time at which a physical resource can be acted on.

We assume throughout that the knowable time arrives at or before the actionable time. The information process is assumed to bring to us the attributes of a resource that might not be actionable. This can be a customer booking an order in advance, or information that a vehicle has departed one location (the information event) headed for a destination in the future (at which point it becomes actionable).

The actionable time can be viewed as nothing more than an attribute of a resource. We can pick one element of the attribute vector a , and give it a name such as $a_{actionable}$. However, it is often useful to explicitly index the actionable time to simplify the process of writing flow conservation constraints, since it will usually be the case that we can only act on a resource (with a decision) that is actionable. For this reason, we let:

$R_{t,t'a}$ = The number of resources that are known at time t to be actionable at time $t' \geq t$ with attribute vector a .

$$R_{tt'} = (R_{t,t'a})_{a \in \mathcal{A}}.$$

$$R_t = (R_{tt'})_{t' \geq t}.$$

Thus, we can continue to use R_t as the resource that we know about at time t . If we write R_{ta} , then we implicitly assume that the actionable time is an attribute. If we write $R_{tt'}$ or $R_{t,t'a}$ then we assume that t' is the actionable time.

The double time index (t, t') provides an explicit model of the information process (the first time index) and the physical process (the second time index). For deterministic models, we could (clumsily) index every variable by R_{0t} , whereas if we want to model processes where the information arrives as the physical event occurs, variables would all be indexed by R_{tt} . Clearly, in both cases it is more natural and compact to use a single time index.

The double time indexing can be used elsewhere. We might plan an activity at time t (implicitly, using the information available at time t) that will happen at time t' in the future. Such a decision vector would be denoted $x_{tt'}$, where x_{tt} is an action (something implemented right away) while $x_{tt'}, t' > t$ would represent a plan of an activity that might happen in the future. We can lock in such plans (normally we might do so for t' within some *decision horizon*), or treat them simply as forecasts of activities in the future (similar to forecasts of customer demands). As we did with R_t , we can let

$$x_t = (x_{tt'})_{t' \geq t}$$

$$c_t = (c_{tt'})_{t' \geq t}.$$

This allows us to retain a compact notation with a single time index, as long as it is understood that these variables can be vectors extending over a range of actionable times in the future.

Although we have not formally introduced our cost function, we can also let $c_{tt'}$ be the cost of an activity planned (using the information available) at time t . We note that the flow of dollars is itself a separate process from the flow of information and physical flows. Normally, payment for a service comes at some time after the service is provided (and as with the flow of the physical resource, there is some uncertainty associated with the financial flows). We are not going to further address the flow of money, but a complete dynamic model would separate the timing of the informational, physical and financial flows.

4.4 Information and Sigma-algebras

Students of mathematical programming learn that they have to understand linear algebra and the meaning of equations such as $Ax = b$. People who work in the area of dynamic information processes (alternatively, stochastic, dynamic models) learn the meaning of statements such as “the function X_t must be \mathcal{F}_t -measurable”. In this section, we undertake a brief explanation of this concept.

We first need to emphasize that measure theory is an abstract part of the field of mathematics that was adopted by the probability community (just as linear algebra was adopted by the math programming community). As a general rule, it is not necessary to have a strong command of measure theory to work successfully in the field of stochastic, dynamic models (in contrast with linear algebra which is essential to work in the field of math programming), but it will add richness to a presentation (and is essential for some theoretical work). Most important is the notion of a concept known as a “sigma-algebra” (often designated σ -algebra) and its role in conveying information. In engineering, information is simply data; to a probabilist, information is always represented as a sigma-algebra.

If the random variables W_t were discrete, we could describe all the elements of Ω as “outcomes,” with $p(\omega)$ the “probability of outcome ω .” In general, this is not the case (random variables can be continuous), in which case we cannot refer to ω as an event. Instead, an event has to be a set of outcomes ω . For the moment (we refine this concept later), let \mathcal{F} be all possible subsets of Ω , with the property that every union of sets in \mathcal{F} is an element of \mathcal{F} , and every complement of \mathcal{F} is a member of \mathcal{F} (the set Ω and the empty set ϕ would both be elements of \mathcal{F}). Since every element of \mathcal{F} is actually a set of events, \mathcal{F} is, literally, a “set of sets.” \mathcal{F} is called a “sigma-algebra.”

It is clear that for every set of outcomes Ω , we can define a sigma-algebra \mathcal{F} , and for every element in \mathcal{F} , we can define a probability. We let \mathcal{P} be a function (which we will write as $p(\omega)$) defined over Ω that allows us to determine the probability of an event in \mathcal{F} (if Ω is discrete, we just have to add up $p(\omega)$ for each ω in an event $\mathcal{E} \in \mathcal{F}$). If there are continuous random variables, then $p(\omega)$ will be a density function that we have to integrate over.

Given the set of outcomes Ω , events \mathcal{F} and a probability measure, we have all the elements of a probability space which we write $(\Omega, \mathcal{F}, \mathcal{P})$. It is not uncommon to find papers which will start by defining a generic probability space $(\Omega, \mathcal{F}, \mathcal{P})$, without actually specifying what these are (nor

do they refer to it again). For engineering applications, the outcomes Ω should be clearly defined in terms of actual information events.

This generic notation applies to any stochastic problem, but we are in particular interested in problems which involve information that arrives over time. Assume that we are at time t (the information during time interval $(t - 1, t)$ has already arrived). We can define a sigma-algebra \mathcal{F}_t consisting of all the events that can be created just using the information available up through time t . This is a fairly subtle concept that is best illustrated by an example. Assume we have a single customer which may or may not make a request in time period t . Our information W_t , then, is the random demand that we might call D_t (if this was our only source of exogenous information, we would just use the variable D_t for our information). In our simple example, D_t can be zero or one. If we are considering three time periods, then our sample space Ω consists of eight potential outcomes, shown in table 1.

Outcome ω	Time period		
	1	2	3
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	1
5	1	0	0
6	1	0	1
7	1	1	0
8	1	1	1

Table 1: Set of demand outcomes

Now assume that we have just finished time interval 1. In this case, we only see the outcomes of D_1 , which can be only 0 or 1. Let $\mathcal{E}_{\{D_1\}}$ be the set of outcomes ω that satisfy some logical condition on D_1 . For example, $\mathcal{E}_{\{D_1=0\}} = \{\omega | D_1 = 0\} = \{1, 2, 3, 4\}$ is the set of outcomes that correspond to $D_1 = 0$. The sigma-algebra \mathcal{F}_1 would consist of the set of sets $\{\mathcal{E}_{\{D_1=0\}}, \mathcal{E}_{\{D_1=1\}}, \mathcal{E}_{\{D_1 \in (0,1)\}}, \mathcal{E}_{\{D_1 \notin (0,1)\}}\}$.

Next assume that we are in time period 2. Now our outcomes consist of the elementary events in $\mathcal{H}_2 = \{(0, 0), (0, 1), (1, 0), (1, 1)\}$, and the sigma-algebra \mathcal{F}_2 would include all the unions and complements of events in \mathcal{H}_2 . For example, one event in \mathcal{F}_2 is $\{\omega | (D_1, D_2) = (0, 1)\} = \{3, 4\}$. Another event in \mathcal{F}_2 is the $\{\omega | (D_1, D_2) = (0, 0)\} = \{1, 2\}$. A third event in \mathcal{F}_2 is the union of these two events, which consists of $\omega = \{1, 2, 3, 4\}$ which, of course, is one of the events in \mathcal{F}_1 . In fact, every event in \mathcal{F}_1 is an event in \mathcal{F}_2 , but not the other way around. The reason is that the additional information from the second time period allows us to divide Ω into a finer set of subsets. Since \mathcal{F}_2 consists of all unions (and complements), we can always take the union of events which is the same

as ignoring some piece of information. By contrast, we cannot divide \mathcal{F}_1 into a finer partition. The extra information in \mathcal{F}_2 allows us to filter Ω into a finer set of subsets than was possible when we only had the information through the first time period. If we are in time period 3, \mathcal{F} will consist of each of the individual elements in Ω , as well as all the unions needed to create the same events in \mathcal{F}_2 and \mathcal{F}_1 .

In other words, additional information allows us to divide our set of outcomes Ω into finer sets of subsets. This is the reason why a sigma-algebra is viewed as representing information.

With each additional time period, the information from that time period allows us to filter Ω into finer and finer subsets. As a result, we can write $\mathcal{F}_1 \subseteq \mathcal{F}_2 \subseteq \dots \subseteq \mathcal{F}_t$. We term this a set of “increasing sub-sigma-algebras.” When this occurs, we term the sequence $(\mathcal{F}_1, \mathcal{F}_2, \dots, \mathcal{F}_t)$ a *filtration*. This is the reason why the notation \mathcal{F}_t is used rather than a more mnemonic notation such as \mathcal{H}_t (as in “history”).

The sequence of sub-sigma-algebras is especially important, even if the mechanism will appear clumsy (not just at first - even experienced professionals feel this way). We can use this notation to specify that a sequential decision process is not cheating by using information in the future. When this is the case, we say that the decision rule is *nonanticipative*. Let X_t be a function that determines a decision at time t (more precisely, using the information available at time t). As further illustration, assume that we are faced with the problem of whether or not to dispatch a single truck with customers or freight waiting to move over a single link. In this case, $X_t \in (0, 1)$, where $X_t = 1$ means to dispatch the truck and $X_t = 0$ means to hold it another time period. We further assume that the function X_t uses some well-defined rule to determine when the vehicle should be dispatched.

Our exogenous information process consists of A_t which is the number of customers arriving to move over our link. This problem is much too complex to use as a numerical example, but all the same concepts apply. However, this time we are going to define a set of events in a more meaningful way. Instead of a sigma-algebra that includes every possible event, we only care about the events that correspond to specific decisions. For example, for the decision X_1 , we might assume that

$$X_1 = \begin{cases} 1 & \text{If } A_1 \geq y \\ 0 & \text{Otherwise.} \end{cases}$$

where y is a prespecified parameter. We do not care about every possible outcome of A_1 ; we only

care about whether it is greater than or equal to y , or less than y . Let $\mathcal{E}_{\{A_1 \geq y\}} = \{\omega | A_1(\omega) \geq y\}$, with $\mathcal{E}_{\{A_1 < y\}}$ defined similarly. We can define a new sigma-algebra which we will call \mathcal{F}_1^X which consists of the events $\{\mathcal{E}_{\{A_1 \geq y\}}, \mathcal{E}_{\{A_1 < y\}}, \Omega, \phi\}$. This is much smaller than \mathcal{F}_1 , but contains the events that are relevant to our decision. We would say that \mathcal{F}_1^X is the sigma-algebra *generated by* the decision function X_1 . Clearly, $\mathcal{F}_1^X \subseteq \mathcal{F}_1$. Although it gets more complicated, we can continue this idea for other time periods. For example, we can think of a sequence of decisions (X_1, X_2, X_3) . Each possible sequence, say $(0, 1, 0)$, corresponds to a set of realizations of (A_1, A_2, A_3) . We can build a set of events \mathcal{F}_3^X around all the possible sequences of decisions.

We have now built two sets of sigma-algebras. The original generic one, \mathcal{F}_t which consists of all the possible events created by the sequence (A_1, \dots, A_t) , and our new one, \mathcal{F}_t^X which identifies the events relevant to our decision function, where $\mathcal{F}_t^X \subseteq \mathcal{F}_t$. Since we have a probability measure \mathcal{P} defined on (Ω, \mathcal{F}) , we can compute the probability of any event in \mathcal{F}_t . This means that we can compute the probability of any event in \mathcal{F}_t^X . When this is the case, we say that “ X_t is \mathcal{F}_t -measurable” because the sigma-algebra generated by the decision function X_t creates sets of events that are already elements of the original sigma-algebra (where we already know the probability of each event). It is common in stochastic optimization to create a decision function X_t and require it to be “ \mathcal{F}_t -measurable.” Equivalent terms are to say that X_t is \mathcal{F}_t -adapted, or that X_t is nonanticipative. All of these mean the same thing: that the decision function has access only to the information that has arrived by time t .

When would this not be the case? Let’s say we cheated, and built X_t in a way that it used A_{t+1} . While this is not physically possible (you cannot use information that has not arrived yet), it is possible in computer simulations where, for example, we are solving a problem that happened in history. We might be running simulations to test new policies on historical data. It is possible in this situation to “cheat,” which in formal terms could be called a “measurability violation.” If a decision was able to see information in the future, we would be able to create a finer grained set \mathcal{F}_t . If some readers feel that this is a clumsy way to communicate a basic idea, they would have a lot of company. But this is fairly standard vocabulary in the stochastic optimization community.

Another way to identify a valid set of decisions is to identify, for each outcome of the exogenous information ω (in our example, the realization of the number of arrivals in each time period), the corresponding set of decisions. Just as we can identify a set of events on the sequence (A_1, A_2, \dots, A_t) , we can define sets of events on the sequence (X_1, X_2, \dots, X_t) . For each $\omega \in \Omega$,

there is a sequence of decisions $(x_1(\omega), x_2(\omega), \dots, x_t(\omega))$. For lack of better notation, let \mathcal{G}_t be the sigma-algebra created by all the possible outcomes of (X_1, X_2, \dots, X_t) . Then, we have a valid information process (that is, no cheating), if $\mathcal{G}_t \subseteq \mathcal{G}_{t+1}$ (that is, it forms a filtration). More than an abstract mathematical concept, the field of stochastic programming uses this relationship to develop an explicit set of constraints to ensure a valid set of decisions (see section 8.2.2).

5 Decisions

Dynamic systems evolve because of information that arrives from outside the system (“exogenous information processes”) and decisions (which can be viewed as “endogenous information processes”). The challenge with decisions is determining how they are made, or how they should be made.

For our presentation, we focus purely on decisions that act on resources (this excludes, for example, pricing decisions). This means that all our decisions are constrained by resources, and their impact on the future is reflected in the resource constraints of future decisions.

5.1 Representing decisions

Although transportation problems are often viewed as consisting of decisions to move from one location to another, real problems also include a variety of other decisions, such as cleaning or repairing equipment, buying/selling equipment, sending crews home on rest, serving customers, loading/unloading equipment, reconfiguring equipment, moving equipment between customer-controlled equipment pools, and so on. To handle this generality we define:

\mathcal{C}^D = Set of decision classes (move to a location, clean/repair, serve a customer request, buy, sell).

\mathcal{D}^c = Set of decision types for class c .

$\mathcal{D} = \bigcup_{c \in \mathcal{C}^D} \mathcal{D}^c$.

It is often necessary, for practical purposes, to define decision sets that depend on the attributes of the resource (which might be a layered resource). For this purpose we let:

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

An element $d \in \mathcal{D}$ is a type of decision. We measure the number of decisions made of each type using:

x_{tad} = The number of resources of type a that a decision d is applied to using the information available at time t .

$$x_t = (x_{tad})_{a \in \mathcal{A}, d \in \mathcal{D}_a}.$$

It is useful to contrast our indexing of decisions. It is common in deterministic models in transportation to use a variable such as x_{tij} to be the flow from i to j departing at time t . In a stochastic model, the outcome of a decision might be random, especially when we are working with vectors of attributes (for example, one attribute might indicate the maintenance status of a piece of equipment). The notation x_{tad} indexes the decision variable by information known when the decision is made, and not on the outcome (the notation for representing the outcome of a decision is given in section 6).

5.2 The decision function

The real challenge, of course, is determining how to make a decision. We assume that we have a decision function that we represent using:

$X_t^\pi(I_t)$ = A function that returns a vector x_t given information I_t .

I_t = The set of functions representing the information available at time t .

Π = A family of decision functions (or policies).

The decision function is required to return a feasible decision vector. We let:

\mathcal{X}_t = The set of feasible decision vectors given the information available at time t .

For example, we would expect \mathcal{X}_t to include constraints such as

$$\sum_{d \in \mathcal{D}_a} x_{tad} = R_{t,ta} \quad \forall a \in \mathcal{A} \tag{7}$$

$$x_{tad} \leq u_{tad} \quad \forall a \in \mathcal{A}, d \in \mathcal{D}_a \tag{8}$$

$$x_{tad} \geq 0 \quad \forall a \in \mathcal{A}, d \in \mathcal{D}_a. \tag{9}$$

	Information class	Algorithmic strategy
1	Knowledge: The data that describes what we know about the system now (myopic models)	Classical deterministic math programming
2a	Forecasts of exogenous information processes (point forecasts)	Rolling horizon procedures
2b	Forecasts of exogenous information processes (distributional forecasts)	Stochastic programming
3	Forecasts of the impacts now on the future (value functions)	Dynamic programming
4	Forecasts of future decisions (patterns)	Proximal point algorithms

Table 2: Four classes of information and corresponding algorithmic strategies

Of course, other constraints might arise in the context of a particular application. We note, however, that it is very common in deterministic models to treat as constraints serving a customer, possibly within a time window. These are rarely expressed as constraints in dynamic models, simply because it is too easy for the model to be infeasible. We encourage limiting \mathcal{X}_t to true physical constraints (such as flow conservation or hard constraints on flow), and let other “desirable behaviors” (such as serving a customer on time) be handled through the objective function.

The information I_t is all the data needed to make a decision, and only the data needed to make a decision. There is, of course, a close relationship between the information available and the type of decision function that is used. In our presentation, it is useful to identify four major classes of decision functions that reflect the type of information that is available. Each class corresponds with a well-established algorithmic strategy. The four classes and corresponding algorithmic strategies are summarized in table 2. The first class uses what we know now, which is a standard myopic model. This covers the vast array of deterministic models which can be solved by algorithms developed for this problem class. This class, of course, is the foundation for the three other classes, which are all assumed to use this information (and these algorithmic strategies), augmented by the information listed for each class.

The second class includes forecasts of future information (this is not events in the future, but rather information that has not yet arrived). When we include information about the future, we can include a point forecast (which is the most widely used) or a distributional forecast. Point forecasts produce the rolling horizon procedures that are widely used in engineering practice (algorithmically, these can be solved using the same techniques as myopic models, but the problems are usually quite a

bit larger). Alternatively, we can use a distributional forecast (where we explicitly represent multiple future scenarios) which puts us in the realm of stochastic programming. These are widely used in financial asset allocation, but have not been successfully applied to problems in transportation and logistics.

The third information class represents a forecast of the impact of a decision now on the future. This puts us into the framework of dynamic programming, where the future impact of a decision now is captured through a value function (also known as a cost-to-go function). A value function can be viewed as the value of being in a particular state in the future, or for some classes, a forecast of a future dual variable.

The fourth and last information class is a forecast of a future decision. This can be accomplished in several ways, but a simple example is of the form “we normally put this type of locomotive on this type of train” or “we normally put our sleeper teams on long loads.” Each of these examples summarizes a behavioral pattern for a company, represented in a form of an aggregated attribute-action pair (it is a type of decision acting on a resource with a subset of attributes).

It is possible, of course, to mix and match algorithmic strategies, but any reasonable strategy would start with the first information class and then seek to improve the solution by adding additional information classes.

6 System dynamics

We next have to model the impact of decisions on the system over time. We represent these effects using a device called the *modify function* which works at the level of an individual resource. The modify function can be represented as the mapping

$$M(t, a, d) \rightarrow (a', c, \tau). \tag{10}$$

Here, we are acting on a resource with attribute vector a with decision type d at time t (which also defines the information available when the decision is implemented). a' is the attribute of the resource after the decision, c is the contribution (or cost), and τ is the time required to complete the decision. It is useful to also write the results of the modify function as functions themselves

$$M(t, a, d) \rightarrow (a^M(t, a, d), c^M(t, a, d), \tau^M(t, a, d)). \tag{11}$$

$a^M(t, a, d)$ is the *attribute transition function*, where the superscript “ M ” is used to help identify the difference between the attribute vector a and the attribute transition function $a^M(t, a, d)$. It is common to use $c_{tad} = c^M(t, a, d)$ and $\tau_{tad} = \tau^M(tad)$.

There are many problems where the evolution of the attribute vector is deterministic. This is particularly true if the only source of randomness is external customer demands. But many problems exhibit behaviors such as equipment failures or random delays due to weather or traffic. For these situations, we need to model the evolution of the attribute vector a_t just as we model the evolution of the state variable S_t . The attribute vector a_t in (11) would be a pre-decision attribute vector. We would change our attribute transition function so that it parallels our transition function, where we might write

$$a_{t+1} = a^M(a_t, d_t, W_{t+1}).$$

where W_{t+1} is the exogenous information that describes equipment failures, delays, or the decision by a customer that a vehicle is dirty or unacceptable.

Any model that manages resources over time would have software that roughly corresponds to the modify function (which might be a single routine or several performing component tasks). The modify function can be arbitrary, rule-based logic, and as such is extremely flexible. However, this is very difficult to use when writing equations, so for this purpose we define

$$\delta_{t,t'a'}(t, a, d) = \begin{cases} 1 & \text{If } a^M(t, a, d) = a' \\ 0 & \text{Otherwise.} \end{cases}$$

The δ function is simply an indicator function that captures the outcome of a decision. We note that it is possible to capture uncertainty in the outcome of a decision. This is one reason why our decision variables x_{tad} are indexed with information known when a decision is made, and not the outcome of the decision. While this somewhat complicates tasks such as summing the flows into a location, it provides an important level of generality (for example, the time required to complete a decision might easily be random).

To model the resource dynamics, we are going to assume that only the vector x_{tt} is implementable (that is, if we are choosing a plan $x_{tt'}, t' > t$, these plans are ignored and replanned in the next time interval). We also assume that x_{tt} can only act on actionable resources (that is, those in

the vector R_{tt} , as we did in equation (7)). Our resource dynamics are given by

$$R_{t+1,t'a'} = R_{t,t'a'} + \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}_a} \delta_{t,t'a'}(t, a, d) x_{tad} + \hat{R}_{t+1,t'a'} \quad t' > t. \quad (12)$$

We note that the right hand side of (12) does not include the case $t = t'$. Resources in the vector R_{tt} are all captured by the decision vector x_t (including the resources where we are doing nothing). Reading across, this equation says: the resources that we will know about at the end of time period $t + 1$ which are actionable at time $t' \geq t + 1$ consist of a) the resources that are actionable at time t' that we knew about at time t (but could not act on), b) the resources that were actionable at time t (we assume we act on everything, even if the action is to “do nothing”), and c) the new resources entering the system during time interval $t + 1$ which are actionable at time t' .

Although it is clearer to write out this equation in this form, it is more compact to write it out in matrix form. Let $\Delta_{tt'}$ be the matrix where $\delta_{t,t'a'}(t, a, d)$ is the element in row a' , column (a, d) . As before, $x_{tt'}$ is a vector with element (a, d) . We can then rewrite (12) as

$$R_{t+1,t'} = R_{t,t'} + \Delta_{tt'} x_{tt'} + \hat{R}_{t+1,t'}. \quad (13)$$

7 An optimization formulation

We are now ready to state our problem in the form of an optimization model. Our problem is to maximize the total contribution received over all time periods. Our contribution function in period t is given by

$$C_t(x_t) = \sum_{d \in \mathcal{D}} \sum_{a \in \mathcal{A}} c_{tad} x_{tad}.$$

We can write our decision model in the general form

$$X_t^\pi(R_t) = \arg \max_{x_t \in \mathcal{X}_t} C_t^\pi(x_t). \quad (14)$$

Note the difference between the actual contributions in a time period, $C_t(x_t)$, and the contributions that we are optimizing over to obtain a set of decisions, $C_t^\pi(x_t)$. Later, we provide specific examples of $C_t^\pi(x_t)$, but for now it suffices to establish the relationship between a policy and the specific

contribution function being used to make a set of decisions. To establish our objective function, we first define

$$F_t^\pi(R_t) = E\left\{\sum_{t'=t}^T C_{t'}(X_{t'}^\pi) \middle| R_t\right\}.$$

$F_t^\pi(R_t)$ is the expected contribution from following policy π starting at time t in state R_t . The best policy is found by solving

$$F_0^*(\mathcal{S}_0) = \sup_{\pi} F_0^\pi(\mathcal{S}_0). \tag{15}$$

We note that we use the supremum operator to handle the mathematical possibility that we are searching over a class of policies that might not be bounded. For example, a policy might be characterized by a parameter that is required to be strictly greater than zero, but where zero produces the best answer. Or, the optimal parameter could be infinite. If the set of policies is contained in a compact set (that is, a set that is closed and bounded), then we can replace the supremum operator by a simple maximization operator.

Equation (15) is computationally intractable, so we generally find ourselves looking at different classes of policies, and finding the best within a class. If this were a deterministic problem (that is, no new information arriving over time), we could write it in the following form

$$\max_x \sum_{t \in \mathcal{T}} c_t x_t \tag{16}$$

subject to

$$A_t x_t - \sum_{t'' < t} \Delta_{t''t} x_{t''} = \hat{R}_t \tag{17}$$

$$x_t \geq 0. \tag{18}$$

In this formulation, the decision variable is the vector of decisions x_t . When the problem is stochastic, it becomes a lot harder to write down the decision variables, because the decisions we make will depend on the state of the system at time t or, more generally, on the particular scenario. For this reason, we cannot simply maximize the expectation, because the expectation is summing over many scenarios, and we can make different decisions for each scenario (this would not be the

case if the decision is a static parameter such as the capacity of a facility or the size of the fleet of vehicles).

An alternative approach is to assume that instead of choosing all the decisions over all the time periods over all the scenarios, that we are going to find a single decision function $X_t^\pi(I_t)$ that returns a decision given the information I_t . The challenge is that this single function has to work well across all the scenarios (which produces different information sets). However, if the function is defined to work only with the information available at time t , then it will automatically satisfy the various measurability/nonanticipativity conditions that we require. Most people will generally find that this approach is more natural.

When we use this approach, our optimization problem looks like

$$\max_{\pi \in \Pi} E \left\{ \sum_{t \in \mathcal{T}^{ph}} c_t X_t^\pi(I_t) \right\}, \quad (19)$$

where \mathcal{T}^{ph} is the set of time periods that define our planning horizon (say, $t = 0, \dots, T$). Unlike classical optimization, we do not specify all the constraints, since we already require that the decision function produce decisions that satisfy the constraints \mathcal{X}_t at a point in time. As a result, we have only to specify constraints that govern the evolution of the system over time, as with equation (12).

We are not going to attempt to solve equation (19) exactly. We quickly give up the search for optimal solutions (or even optimal policies) if we can find “good” policies that work well over as many scenarios as possible. Such policies are often referred to as *robust*. In the next section, we address the problem of finding good policies.

8 Algorithmic strategies

Unlike deterministic optimization problems where there is a well defined concept of an optimal solution (in the case of integer programs, these can be difficult to both find and identify, but at least they are well defined). In the dynamic world, there will be valid disagreements over what even constitutes a valid policy.

In this section, we summarize four major classes of policies, and offer these as choices for modelers to consider and adapt to their needs. These are complementary strategies, since they can

be used in various combinations depending on the characteristics of the problems at hand. Each policy is going to use a new class of information, and open up a new algorithmic strategy. But all the strategies start with the first strategy.

8.1 Myopic models

The most basic strategy on which all other methods build is a myopic policy, which optimizes over information that is known now, but extends into the future (we might take a driver available three hours from now and assign him to a load that will be available five hours from now). The resulting model can be stated as

$$\max_{x_0} \sum_{t' \in \mathcal{T}^{ph}} c_{0t'} x_{0t'} \quad (20)$$

subject to

$$A_{00}x_{00} = R_{00} \quad (21)$$

$$A_{0t'}x_{0t'} - \sum_{t''=0}^{t'-1} \Delta_{t''t'} x_{0t''} = \hat{R}_{0t'} \quad t' \geq 1 \quad (22)$$

$$x_{0t'} \geq 0 \quad \text{and possibly integer.} \quad (23)$$

Here, $x_{0t''}$ is a vector of decisions to be implemented at time t'' using the information available at time 0. The matrix $\Delta_{t''t'}$ is an indicator matrix that tells us when a decision implemented at time t'' produces a resource that is actionable at time t' . Technically, we need a third time index to show that $\Delta_{t''t'}$ is computed with information available at time 0.

With this generic model we include the entire range of classical optimization problems and algorithms that have been studied in transportation and logistics. These might be simple assignment problems, vehicle routing problems, network design problems and multicommodity flow problems. In general, however, these are much smaller than the types of problems that arise in planning applications. We assume that this basic myopic problem is solvable to an acceptable level of optimality. This is the fundamental building block for all other modeling and algorithmic strategies.

There are some problems where it is possible to provide effective solutions using a simple myopic model. Assigning taxi drivers to customers, assigning long-haul truckload drivers to loads, and the

dynamic routing and scheduling of pickup and delivery trucks in a city. But our experience has been that while these models can be effective, they are always missing something.

8.2 Rolling horizon procedures

We can extend our basic myopic model by including information that is forecasted to come in the future. A common procedure is to use a point forecast in a rolling horizon procedure (section 8.2.1), but we can also use a distributional forecast in a stochastic programming algorithm (section 8.2.2).

8.2.1 Deterministic models

The most common approach to combining current knowledge with forecasted information is to use a point forecast of future events which produces a classic rolling horizon procedure using point forecasts. Let $\bar{R}_{t'}$ be the forecast of information that becomes available during time interval t that is actionable at time t' . We note that these forecasts are known at time 0. Using these forecasts, our optimization problem becomes

$$\max_{x_0} \left(\sum_{t \in \mathcal{T}^{ph}} (c_{0t} x_{0t}) + \sum_{t' \in \mathcal{T}^{ph} \setminus 0} c_{t'} x_{0t'} \right) \quad (24)$$

subject to, for $t \in \mathcal{T}^{ph}$

$$A_{00} x_{00} = R_{00} \quad (25)$$

$$A_{0t} x_{0t} - \sum_{t'=0}^{t-1} \Delta_{t't} x_{0t'} = \sum_{t'=0}^t \bar{R}_{t't} \quad t > 0 \quad (26)$$

$$x_{0t} \geq 0. \quad (27)$$

Note that the right hand side of equation (26) is the total number of resources that are actionable at time t' . To obtain this, we have to sum over all the information that is forecasted to arrive in future time periods. It is interesting that many practitioners view this as the “correct” way to solve dynamic problems. They recognize that point forecasts are not perfect, but argue that you simply reoptimize as new information comes in.

Rolling horizon procedures have the primary benefit that they use classical modeling and algorithmic technologies. In addition, point forecasts are the easiest to understand in the business

community. But as with myopic models, they will over time start to show their weaknesses. For example, rail customers often exhibit highly variable demands. A forecasting system will tend to forecast the average demand, while a seasoned operations manager will know that you need to supply the customer more than the average to cover the days when the demand is high. Waiting until the last minute might not provide enough time to move cars to satisfy the demand.

Another problem class where point forecasts perform poorly are discrete routing and scheduling. Here, a customer demand will normally be zero but will sometimes be one. A point forecast would produce a fraction such as 0.2. It is not possible to route an entire vehicle to each customer with a nonzero (fractional) demand forecast. Randomly sampling the demands is nothing more than routing to a random outcome. Allowing a model to send a fraction of a vehicle to a fraction of a demand will create an extremely difficult integer program for the first period.

8.2.2 Stochastic programming

If we wish to explicitly model the possibility of multiple outcomes, we find ourselves in the domain of stochastic programming. Now, we have to allow for the possibility that different decisions will be made in the future depending on the actual evolution of information.

In this case, our optimization problem becomes

$$\max_x \left(\sum_{t \in \mathcal{T}^{ph}} (c_{0t}x_{0t}) + \sum_{\omega \in \hat{\Omega}} \hat{p}(\omega) \sum_{t \in \mathcal{T}^{ph} \setminus \{0\}} \sum_{t' > t} c_{tt'}x_{tt'}(\omega) \right). \quad (28)$$

subject to, for $t, t' \in \mathcal{T}^{ph}$:

First stage constraints:

$$A_{00}x_{00} = R_{00} \quad (29)$$

$$A_{0t'}x_{0t'} - \sum_{t''=0}^{t'-1} \Delta_{t''t'}x_{0t''} = R_{0t'} \quad t' > 0 \quad (30)$$

$$x_{0t'} \geq 0 \quad t' \geq t. \quad (31)$$

Later stage constraints, for all $\omega \in \hat{\Omega}, t, t' \in \mathcal{T}^{ph} \setminus 0, t' \geq t$:

$$A_{tt}(\omega)x_{tt}(\omega) = R_{tt}(\omega) \quad (32)$$

$$A_{tt'}(\omega)x_{tt'}(\omega) - \sum_{t''=t}^{t'-1} \Delta_{t''t'}(\omega)x_{tt''}(\omega) = \hat{R}_{tt'}(\omega) \quad (33)$$

$$x_{tt'}(\omega) \geq 0. \quad (34)$$

This formulation requires that we choose a vector of decisions for the first time period x_{0t} but allows a different set of decisions for each scenario in the later time periods. This might not be a bad approximation, but it does allow the decisions in later time periods to “see” future information. This happens because we are indexing a decision by an outcome ω which implicitly determines all future information. In the vocabulary of stochastic programming, we prevent this by adding what are known as nonanticipativity constraints. We can express these by using some of our notation (and concepts) from section 4.4. As before, let $h_t = (\omega_1, \omega_2, \dots, \omega_t)$ be the history of the process up through time t , and let \mathcal{H}_t be the set of all possible histories. In addition, define the subset $\Omega_t(h_t) = \{\omega \in \hat{\Omega} | (\omega_1, \omega_2, \dots, \omega_t) = h_t\}$ to be the set of scenarios $\omega \in \hat{\Omega}$ which match the history h_t up through time t . We would then add the constraint

$$x_t(h_t) = x_t(\omega) \quad \forall \omega \in \Omega_t(h_t), \quad h_t \in \mathcal{H}_t. \quad (35)$$

Equation (35) is our nonanticipativity constraint, which has the effect of forcing the set of decisions $(x(\omega), \omega \in \hat{\Omega})$ to form a filtration. Literally, we want each set of decisions with the same history to be the same.

For most transportation problems, the optimization problem (28)-(35) is too large to be solved, although special cases exist (see, for example, Morton et al. (2002)), Glockner & Nemhauser (2000)).

8.3 Dynamic programming

A powerful technique for taking into account uncertain future events is dynamic programming, which uses the construct of a value function to capture the expected value of being in a particular state in the future. If we can let R_t represent the state of our system, we can calculate the value $V_t(R_t)$ of being in state R_t from time t onward using the well-known Bellman equations

$$V_t(R_t) = \max_{x_t \in \mathcal{X}_t} \left(C_t(x_t) + \sum_{R' \in \mathcal{R}} P(R'|R_t, x_t) V_t(R') \right) \quad (36)$$

where $P(R'|R_t, x_t)$ is the probability of being in state R' if we are in state R_t and take action x_t . A more general way of writing equation (36) (and more useful for our purposes here) is using

$$V_t(R_t) = \max_{x_t \in \mathcal{X}_t} [C_t(x_t) + E\{V_{t+1}(R_{t+1})|R_t\}], \quad (37)$$

where $R_{t+1} = R_t + \Delta_t x_t + \hat{R}_{t+1}$. It is generally assumed that we are going to compute equation (36) for each possible value of R_t . As we computed in equation (1), the number of possible values of R_t is extremely large even for fairly small problems. This is the well-known ‘‘curse of dimensionality’’ that has been recognized since dynamic programming was first discovered. For our problem class, there are actually three curses of dimensionality: the state space, the outcome space, and the action space. Since new information (such as \hat{R}_t) is typically a vector, sometimes of high dimensionality, the expectation in equation (37) will generally be computationally intractable. The more conventional form of Bellman’s equation, shown in equation (36) hides this problem since the one-step transition matrix is itself an expectation. Finally, if we compute $V_t(R_t)$ for each discrete value of R_t , then the only way to find the optimal x_t is to search over each (discrete) element of the set \mathcal{X}_t . Again, if x_t is a vector (for our problems, the dimensionality of x_t can easily be in the thousands or tens of thousands) the size of \mathcal{X}_t is effectively infinite.

The dynamic programs depicted in equations (36) and (37) assume a particular representation of states, information and action. We can write the *history* of our process up through time t as

$$h_t = (R_0, x_0, \omega_1, R_1, x_1, \dots, \omega_t, R_t, x_t). \quad (38)$$

The evolution of the state variable is often depicted using

$$R_{t+1} = R^M(R_t, x_t, \omega_{t+1}). \quad (39)$$

where $R^M(\cdot)$ captures the dynamics of the system. We can write the decision function as:

$$X_t^\pi(R_t) = \arg \max_{x_t \in \mathcal{X}_t} (c_t x_t + E(V_{t+1}(R_{t+1})|R_t)). \quad (40)$$

Since the expectation is typically computationally intractable, we can replace this with an approximation. Let $\hat{\Omega} \subset \Omega$ be a small set of potential outcomes. This would be written

$$X_t^\pi(R_t) = \arg \max_{x_t \in \mathcal{X}_t} \left(c_t x_t + \sum_{\hat{\omega} \in \hat{\Omega}} \hat{p}(\hat{\omega}) V_{t+1}(R_{t+1}(\hat{\omega})) \right). \quad (41)$$

where $\hat{p}(\hat{\omega})$ is the probability of the sampled outcome $\hat{\omega} \in \hat{\Omega}$. The problem we encounter in practice is that even when $\hat{\Omega}$ is relatively small, it can still greatly complicate solving (41). The problem we encounter in transportation and logistics is that the myopic problem

$$X_t^\pi(R_t) = \arg \max_{x_t \in \mathcal{X}_t} c_t x_t \quad (42)$$

can be computationally difficult. This could be a vehicle routing problem or integer multicommodity flow problem. Adding the summation over multiple outcomes (as we do in equation (41)) can make a hard problem much larger and harder. We could avoid this by using a single sample $\omega_{t+1} = W_{t+1}(\omega)$, producing

$$X_t^\pi(R_t, \omega) = \arg \max_{x_t \in \mathcal{X}_t} (c_t x_t + V_{t+1}(R_{t+1}(\omega_{t+1}))). \quad (43)$$

Now we are choosing x_t given $R_{t+1}(\omega_{t+1})$ which is like making a decision now given the information that will arrive in the next time period. This will not even be a good approximation.

We can overcome this problem by introducing the concept of measuring the state of the system immediately *after* a decision is made (or equivalently, before new information has arrived). We refer to our original state variable R_t as the “pre-decision” state variable, and we introduce a new state variable, denoted R_t^x as the “post-decision” state variable. We can now write the history of our process as

$$h_t = (R_0, x_0, R_0^x, \omega_1, R_1, x_1, R_1^x, \dots, \omega_t, R_t, x_t, R_t^x). \quad (44)$$

If we write the evolution of our system in terms of R_t^x , we obtain

$$R_t^x = R_t^M(R_{t-1}^x, \omega_t, x_t). \quad (45)$$

Writing the Bellman equations around R_t^x we obtain

$$V_{t-1}^x(R_{t-1}^x) = E \left\{ \max_{x_t \in \mathcal{X}_t} (c_t x_t + V_t^x(R_t^x(x_t))) | R_{t-1}^x \right\}. \quad (46)$$

The relationship between the pre- and post-decision value functions is best illustrated using

$$\begin{aligned} V_t(R_t) &= \max_{x_t \in \mathcal{X}_t} c_t x_t + V_t^x(R_t^x(x_t)) \\ V_{t-1}^x(R_{t-1}^x) &= E \{ V_t(R_t) | R_{t-1}^x \}. \end{aligned}$$

Using equation (46) we still encounter the problem of computing the expectation, but now consider what happens when we drop the expectation and solve it for a single sample realization ω

$$X_t^\pi(R_{t-1}, \omega_t) = \arg \max_{x_t \in \mathcal{X}_t(\omega)} c_t x_t + V_t^x(R_t^x(x_t)). \quad (47)$$

Here, we are finding the decision x_t given the information $\omega_t = \hat{R}_t(\omega)$ which would be available when we make a decision. As a result, we are not violating any informational (measurability or anticipativity) constraints. What is most important is that $V_t^x(R_t^x(x_t))$ is a deterministic function of x_t , so we do not have to work with even an approximation of an expectation (the approximation is implicit in $V_t^x(R_t^x(x_t))$).

We have now crossed a major hurdle, but we still have the problem that we do not know $V_t^x(R_t^x)$. We are going to need to use an approximation which will be estimated iteratively. At iteration n , we would solve (47) using the approximation from iteration $n - 1$

$$X_t^\pi(R_{t-1}, \omega_t^n) = \arg \max_{x_t \in \mathcal{X}_t(\omega^n)} c_t x_t + \bar{V}_t^{n-1}(R_t^x(x_t)) \quad (48)$$

where we drop the superscript x when writing the approximation \bar{V} . We denote the function produced by solving the right hand side of (48) by

$$\tilde{V}_t^n(R_t^n(\omega^n)) = \max_{x_t \in \mathcal{X}_t(\omega^n)} c_t x_t + \bar{V}_t^{n-1}(R_t^x(x_t)). \quad (49)$$

A brief note on notation is in order here. $\tilde{V}_t^n(R_t^n(\omega^n))$ is an approximation of $V_{t-1}^x(R_{t-1}^x)$, which suggests an inconsistency in how we are indexing by time (V^x is indexed by t , while \tilde{V} is indexed by $t - 1$). Recall that our time indexing indicates the information content. V_{t-1}^x is an expectation conditioned on R_{t-1}^x , and therefore contains information up through $t - 1$. $\tilde{V}_t^n(R_t^n(\omega^n))$ is a conditional approximation, computed using the information that arrives in time t . It does not represent even an approximation of an expectation (by contrast $\bar{V}_t^n(R_t)$ is an approximation of the expected value function conditioned on R_t).

We use the information from solving (49) to update the value function approximation. How this update is performed depends on the type of approximation used, so for now we represent this updating process simply as

$$\bar{V}_{t-1}^n = U^V(\bar{V}_{t-1}^{n-1}, \tilde{V}_t^n, R_t^n(\omega^n)). \quad (50)$$

The algorithm works by stepping forward through time using the value function approximation from the previous iteration. This is the standard approach used in the field of approximate dynamic programming. Instead of finding the value function over all possible states, we develop approximations around the states that are actually visited. The two textbooks that describe these techniques (Bertsekas & Tsitsiklis (1996) and Sutton & Barto (1998)) both describe these procedures in the context of steady state problems. Operational problems in transportation and logistics require time-dependent formulations.

There are two strategies for updating the value function (especially when a time-dependent formulation is used). The first uses a single, forward pass, updating the value function approximations as we proceed. The steps of this procedure are outlined in figure 3. The second uses a two-pass procedure, stepping forward through time determining states and actions, and a backward pass that updates the value function using information about the states that were actually visited. The steps of this procedure are given in figure 4.

STEP 0: Initialization:

Initialize \bar{V}_t^0 , $t \in \mathcal{T}$.

Set $n = 0$.

Initialize R^0 .

STEP 1: Do while $n \leq N$:

Choose $\omega = (\omega_1^n, \omega_2^n, \dots, \omega_T^n)$

STEP 2: Do for $t = 0, 1, \dots, T$:

STEP 2a: Solve equation (49) to obtain $\tilde{V}_t^n(R_t^n)$ and x_t^n .

STEP 2b: Compute $R_t^{x,n} = R_t^M(R_{t-1}^n, \omega_t^n, x_t^n)$.

STEP 2c: Update the value function approximations using $\bar{V}_{t-1}^n \leftarrow U^V(\bar{V}_{t-1}^{n-1}, \tilde{V}_t^n, R_{t-1}^{x,n})$.

STEP 3: Return the policy $X_t^\pi(R_t, \bar{V}_t^N)$.

Figure 3: Single pass version of the approximate dynamic programming algorithm

In our applications, we will generally be more interested in derivatives than the actual value function itself. When we solve the approximation $\tilde{V}_t^n(R_t)$ in equation (49), we will typically be solving these problems subject to flow conservation constraints on the resources as in equation (7). As a general rule our problems are continuous but nondifferentiable in R_t . If we are solving linear resource allocation problems, we can obtain stochastic subgradients just by using the dual variable for the flow conservation constraint. In general, these dual variables are subgradients of $\tilde{V}_t^n(R_t)$ only, which depends on $\bar{V}_t^{n-1}(R_t(x_t))$. These can be effective, but the use of approximations from

STEP 0: Initialize \bar{V}_t^0 , $t \in \mathcal{T}$.
Set $n = 0$.
Initialize R^0 .

STEP 1: Do while $n \leq N$:
Choose $\omega = (\omega_1^n, \omega_2^n, \dots, \omega_T^n)$

STEP 2: Do for $t = 0, 1, \dots, T - 1$:
STEP 2a: Solve equation (49) to obtain $\hat{V}_t^n(R_t)$ and x_t^n .
STEP 2b: Compute $R_t^{x,n} = R_t^M(R_{t-1}^{x,n}, \omega_t^n, x_t^n)$

STEP 3: Do for $t = T - 1, T - 2, \dots, 1, 0$:
STEP 3a: Recompute $\tilde{V}_t^n(R_t^n)$ using \bar{V}_{t+1}^n and the decision x_t^n from the forward pass:

$$\tilde{V}_t^n(R_t^n) = c_t x_t + \bar{V}_{t+1}^n(R_{t+1}^n) \tag{51}$$

STEP 3b: Update the value function approximations, $\bar{V}_{t-1}^n \leftarrow U^V(\bar{V}_{t-1}^{n-1}, \tilde{V}_t^n, R_{t-1}^{x,n})$.

STEP 4: Return policy $X_t^\pi(R_t, \bar{V}^N)$.

Figure 4: Double pass version of the approximate dynamic programming algorithm

previous iterations will slow the overall convergence.

Ideally, we would prefer to have gradients of all future profits. Let

$$V_t^\pi(R_t, \omega) = \sum_{t'=t}^T c_{t'} X_{t'}^\pi(R_{t'}, \omega) \tag{52}$$

be the total future profits given an initial resource vector R_t , and sample realization ω , under policy π . We can obtain the gradient of $V_t^\pi(R_t, \omega)$ by storing the basis obtained as we step forward in time. The basis will allow us to identify the effect of, say, adding one more unit to an element R_{ta} by giving us the impact on costs in time period t , and the impact on future resources R_{t+1} . By storing this information as we step forward in time, we can then compute the effect of increasing R_{ta} by one by stepping backward through time, combining the impact of an additional unit of resource of each type at each time $t' \geq t$ with the value of one more unit of resource at the next time period.

8.4 Incorporating low-dimensional patterns

In practice, it is common to carefully develop a math programming-based optimization model, only to find that a knowledgeable expert will criticize the behavior of the model. It is possible, although generally unlikely, that the expert is simply finding a better solution than the algorithm. More

often, the expert simply recognizes behavior that will incur a cost that is not captured by the model, or expresses a preference for an alternative behavior because of other unquantified benefits.

One response to such input is to “fix the model” which means improving the cost function or adding constraints that capture the physical problem more realistically. For some problems, this is the only solution. However, it can often take weeks or months to implement these fixes, which might easily require data that was not originally available. Often, the complaints about the model can be expressed in fairly simple ways:

- You should avoid sending that type of cargo aircraft into Saudi Arabia since we do not have the ability to handle certain types of repairs.
- Do not send Cleveland-based drivers into Indianapolis because their union is in a different conference and it creates a lot of bookkeeping problems.
- You should put your sleeper teams on longer loads so that you get higher utilization out of the two drivers.
- Do not use the six-axle locomotives on certain trains out of Denver because the track is too curved to handle the long locomotives.
- Try to send flatcars that can handle trailers into Chicago because that location moves a lot of trailers from trucking companies.
- Avoid sending drivers domiciled in Florida into Texas because we do not have very much freight out of that location that terminates in Florida, making it hard to get drivers home.
- Avoid putting inexperienced drivers on service-sensitive loads.

All of these preferences can be stated as low dimensional patterns, each of which consists of three elements: the attributes of the resource being acted on at some level of aggregation, the decision being made (also at some level of aggregation), and the fraction of time that the pattern should occur. Aggregation on the attributes arises because each rule is typically trying to accomplish a particular objective, such as getting a driver home or avoiding sending equipment needing maintenance into areas that do not have the proper repair facilities. A pattern measures how often a particular decision is applied to a group of resources, and the decisions will also generally be expressed at an aggregate level. For example, we are not specifying a rule that a type of driver

should (or should not) be assigned to move a particular load of freight. Instead, we are expressing preferences about types of loads (high priority, loads into a particular destination). Finally, these “rules” are generally not hard constraints, but rather expressions of behaviors to be encouraged or discouraged.

Each pattern can be thought of as a preference about the desirability of a particular type of state action pair, which can be represented as aggregations on the attribute vector and decision. For this, we define:

\mathcal{P} = A collection of different patterns.

G_a^p = Aggregation function for pattern $p \in \mathcal{P}$ that is applied to the attribute vector a .

G_d^p = Aggregation function for pattern $p \in \mathcal{P}$ that is applied to the decision d .

These aggregation functions create new attribute spaces and decision sets

$$\begin{aligned}\mathcal{A}^p &= \{G_a^p(a) | a \in \mathcal{A}\} \\ \mathcal{D}_a^p &= \{G_d^p(d) | d \in \mathcal{D}_a\}.\end{aligned}$$

For notational compactness, it is useful to let \bar{a} represent a generic aggregated attribute vector, and \bar{d} an aggregated decision. We use this notation when the level of aggregation is either not relevant, or apparent from the context. A pair (\bar{a}, \bar{d}) represents an aggregated state/action pair (literally, an attribute vector/action pair). The last element of a pattern is the fraction of time that we expect to observe decision \bar{d} when we are acting on a resource with attribute \bar{a} , which we represent as follows:

$\rho_{\bar{a}\bar{d}}^p$ = The fraction of time that resources with attributes $\{a | G_a^p(a) = \bar{a}\}$ are acted on using decisions $\{d | G_d^p(d) = \bar{d}\}$.

Given a flow vector $x_t = (x_{tad})_{a \in \mathcal{A}, d \in \mathcal{D}}$ returned by the model, the number of times that the flow matches a pattern is given by

$$\begin{aligned}\bar{x}_{t\bar{a}\bar{d}}^p &= G^p(x) \\ &= \sum_{\forall a \in \mathcal{A}} \sum_{\forall d \in \mathcal{D}_a} x_{tad} I_{\{G_a^p(a) = \bar{a}\}} I_{\{G_d^p(d) = \bar{d}\}}\end{aligned}\tag{53}$$

where $I_X = 1$ if X is true. We also need to express the resource state variable R_t in an aggregated form

$$\begin{aligned}\bar{R}_{t\bar{a}}^p &= \sum_{\forall a \in \mathcal{A}} R_{ta} I_{\{G_a^p(a)=\bar{a}\}} \\ &= \text{The number of resources with attribute } \bar{a} \text{ at time } t \\ R_{\bar{a}}(x) &= \sum_{t \in \mathcal{T}} \sum_{\bar{d} \in \mathcal{D}_{\bar{a}}^p} x_{t\bar{a}\bar{d}}.\end{aligned}$$

Finally, we define the model pattern flows as

$$\begin{aligned}\rho_{t\bar{a}\bar{d}}(x) &= \frac{x_{t\bar{a}\bar{d}}}{R_{t\bar{a}}} \quad \forall \bar{a} \in \mathcal{A}^p, \quad \forall \bar{d} \in \mathcal{D}_{\bar{a}}^p \quad \forall t \in \{1, 2, \dots, T\} \\ &= \text{The fraction of time that resources with attribute } \bar{a} \text{ are acted on with decision } \bar{d} \text{ at time } t. \\ \rho_{\bar{a}\bar{d}}(x) &= \frac{\sum_{t=1}^T x_{t\bar{a}\bar{d}}}{R_{\bar{a}}} \quad \forall \bar{a} \in \mathcal{A}^p, \quad \forall \bar{d} \in \mathcal{D}_{\bar{a}}^p \\ &= \text{The fraction of time that resources with attribute } \bar{a} \text{ are acted on with decision } \bar{d} \text{ over the entire horizon.}\end{aligned}$$

The static flow fraction after iteration n can also be written as

$$\begin{aligned}\rho_{\bar{a}\bar{d}}^n &= \frac{\sum_{t \in \mathcal{T}} x_{t\bar{a}\bar{d}}^n}{R_{\bar{a}}^n} \\ &= \sum_{t \in \mathcal{T}} \frac{x_{t\bar{a}\bar{d}}^n}{R_{t\bar{a}}^n} \frac{R_{t\bar{a}}^n}{R_{\bar{a}}^n} \\ &= \sum_{t \in \mathcal{T}} \left(\rho_{t\bar{a}\bar{d}}^n \frac{R_{t\bar{a}}^n}{R_{\bar{a}}^n} \right).\end{aligned}\tag{54}$$

Our goal is to create a solution \bar{x}^p that reasonably matches the exogenous pattern ρ^p . We do this by introducing the metric

$$H(\rho(x), \rho) = \sum_{p \in \mathcal{P}} \sum_{\bar{a} \in \mathcal{A}^p} \sum_{\bar{d} \in \mathcal{D}_{\bar{a}}^p} R_{\bar{a}\bar{d}} \left(\rho_{\bar{a}\bar{d}}(x) - \rho_{\bar{a}\bar{d}}^p \right)^2.\tag{55}$$

We often refer to $H(\rho(x), \rho)$ as the ‘‘happiness function’’ since the domain expert evaluating the model tends to be ‘‘happier’’ when the model flows reasonably match the exogenous pattern. More formally, $H(\rho(x), \rho)$ is a well-defined distance matrix used in proximal point algorithms to produce solutions. We now wish to solve

$$\max_{\pi \in \mathcal{P}} \sum_{t \in \mathcal{T}} (c_t X_t^\pi - \theta H(\rho(x), \rho))\tag{56}$$

where θ is a scaling parameter (these might depend on the specific pattern). Our decision function at time t is given by

$$X_t^\pi = \arg \max_{x_t \in \mathcal{X}_t} (c_t x_t - \theta H(\rho(x), \rho))$$

We are generally unable to solve this problem, so we use instead sequences of approximations of the pattern metric, which for the moment we can write as

$$X_t^\pi = \arg \max_{x_t \in \mathcal{X}_t} \left(c_t x_t - \theta \hat{H}^{n-1}(\rho^{n-1}(x), \rho) \right)$$

We encounter an algorithmic challenge because the proximal term is defined over time but we have to solve the problem one time period at a time. We overcome this problem by solving a series of separable, quadratic approximations of the pattern metric. We start by computing the gradient of the pattern metric, which we denote $\hat{H}^n(x, \rho)$, by differentiating (55)

$$\begin{aligned} \hat{H}_{\bar{a}\bar{d}}^n(\rho) &= \left. \frac{\partial H}{\partial \rho_{\bar{a}\bar{d}}} \right|_{\rho_{\bar{a}\bar{d}} = \rho_{\bar{a}\bar{d}}^n} \quad \forall \bar{a} \in \mathcal{A}^p, \quad \forall \bar{d} \in \mathcal{D}_{\bar{a}}^p \\ &= 2R_{\bar{a}}^n(\rho_{\bar{a}\bar{d}}^n - \rho_{\bar{a}\bar{d}}) \quad \forall \bar{a} \in \mathcal{A}^p, \quad \forall \bar{d} \in \mathcal{D}_{\bar{a}}^p \end{aligned} \quad (57)$$

where p is the pattern associated with the level of aggregation in \bar{a} and \bar{d} .

A convergent algorithm can be designed using a Gauss-Seidel strategy which optimizes the pattern metric computed by using flows $x_{t'}^n$ for $t' < t$, and $x_{t'}^{n-1}$ for $t' > t$, and then optimizing x_t . We then solve a separable, quadratic programming approximation at each point in time.

To describe the algorithm in detail, we first define

$$R_{\bar{a}}^n(t) = \sum_{t'=1}^{t-1} \sum_{\bar{d} \in \mathcal{D}_{\bar{a}}^p} x_{t'\bar{a}\bar{d}}^n + \sum_{t'=t}^T \sum_{\bar{d} \in \mathcal{D}_{\bar{a}}^p} x_{t'\bar{a}\bar{d}}^{n-1}$$

and

$$\tilde{\rho}_{\bar{a}\bar{d}}^n(t) = \sum_{t'=1}^{t-1} \left(\frac{x_{t'\bar{a}\bar{d}}^n}{R_{\bar{a}}^n(t')} \right) + \sum_{t'=t}^T \left(\frac{x_{t'\bar{a}\bar{d}}^{n-1}}{R_{\bar{a}}^n(t')} \right) \quad \forall \bar{a} \in \mathcal{A}, \quad \forall \bar{d} \in \mathcal{D}_{\bar{a}}^p, \quad \forall t \in \{1, 2, \dots, T\} \quad (58)$$

The Gauss-Seidel version of the gradient is now

$$\tilde{h}_{t\bar{a}\bar{d}}^n = 2R_{\bar{a}}^n(t)(\tilde{\rho}_{\bar{a}\bar{d}}^n(t) - \rho_{\bar{a}\bar{d}}) \quad \forall \bar{a} \in \mathcal{A}, \quad \forall \bar{d} \in \mathcal{D}_{\bar{a}}^p \quad \forall t \in \{1, 2, \dots, T\}. \quad (59)$$

Our algorithm proceeds by iteratively solving subproblems of the form

$$x_t^n = \arg \max \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}_a} c_{tad} x_{tad} - \theta \sum_{p \in \mathcal{P}} \sum_{\bar{a} \in \mathcal{A}^p} \sum_{\bar{d} \in \mathcal{D}_{\bar{a}}^p} \left(\frac{1}{R_{\bar{a}}^n(t)} \sum_{\bar{d} \in \mathcal{D}_{\bar{a}}^p} \int_0^{x_{t\bar{a}\bar{d}}} [\tilde{h}_{t\bar{a}\bar{d}}^n + 2(u - x_{t\bar{a}\bar{d}}^{n-1})] du \right). \quad (60)$$

This approach has been found to produce solutions that closely match exogenous patterns within as few as two or three iterations.

The real value of exogenous patterns is that it allows an expert to change the behavior of the model quickly by editing an external data file. This approach does not replace the need to ensure the best quality model through careful engineering, but it does allow us to produce more realistic behaviors quickly by simply changing an external data file.

8.5 Summary: the four information classes

This section has outlined a series of modeling and algorithmic strategies that can be largely differentiated based on the information being made available to the model. We started in section 8.1 by providing a basic myopic model that only uses the *information* available at time t . There was no attempt to incorporate any information that had not yet arrived. All the remaining models use some sort of forecast. Section 8.2 illustrates the use of point and distributional forecasts of future information, which produces classical rolling horizon procedures. Section 8.3 introduces a forecast of the value of resources in the future, represented as a value function. This information brings into play the theory of dynamic programming. Finally, section 8.4 uses what might be called a forecast of a decision, expressed in the form of low-dimensional patterns. As an alternative, we could have avoided mathematical optimization entirely and resorted instead to an extensive set of rules; this would be the approach that the artificial intelligence or simulation communities would have adopted. For transportation problems, the rules simply become too complex. Instead, we have found that expert knowledge can often be represented in the form of low dimensional patterns, and incorporated using a proximal point term, bringing into play the theory of proximal point algorithms.

Up to now, we have represented our decision functions as depending on the resource vector R_t . In practice, what we know at time t can also include other data such as estimates of system

parameters such as travel times and costs, and functions such as forecasting models. It is useful to summarize “what we know now” as consisting of data knowledge (such as the resource vector) and functional knowledge (such as models for forecasting demand). Let ν_t be our estimates at time t of various parameters, such as those that govern the modify function. Our data knowledge at time t is then given by:

$$K_t = (R_t, \nu_t)$$

We differentiate between a forecast model (which we know now) from an actual forecast of an event in the future. A forecast is an actual generation of information that might arrive in the future. Let:

$\omega_{[t, T^{ph}]}$ = A vector of potential information events over the planning horizon $(t, t+1, \dots, T^{ph})$.

$\Omega_{[t, T^{ph}]}$ = The set of all potential information events that we wish to consider over the planning horizon.

We feel that it is important to recognize that $\bar{V}_t(R_t)$ is a forecast, using information available up through time t (just as we would from our demand forecast) of the value of resources in the future. Furthermore, we feel that $\bar{V}_t(R_t)$ is a sufficiently different type of forecast that it should be recognized as an information class distinct from Ω_t .

Finally, we feel that when an expert in operations expresses opinions about how a system should behave (represented using the pattern vector ρ_t), this is also a form of forecast that brings into play the experience of our expert. ρ_t can be thought of as a forecast of a decision that the model should make. It will not be a perfect forecast, as we will not in general be able to match these patterns perfectly, but just as we try to match historical patterns using a demand forecasting routine, these patterns are a forecast which reflects patterns of past decisions.

We have, now, four classes of information

K_t = The data knowledge at time t .

$\Omega_{[t, T^{ph}]}$ = A set of potential outcomes of exogenous information events in the future.

$\bar{V}_t(R_t)$ = Forecasts of the impact of decisions now on the future.

ρ_t = Forecasts of future decisions, captured in the form of low-dimensional patterns.

We can now formulate an information set I_t which will always include K_t , and can also include any combination of the remaining information sets. If we are solving the problem using only a rolling horizon procedure, we would specify $I_t = (K_t, \Omega_{[t, T^{ph}]})$. If we want to use dynamic programming, we would write $I_t = (K_t, \bar{V}_t(\Omega_t))$ to express the fact that our decision function depends on value functions which themselves depend on forecasts of future events. All four information classes would be written: $I_t = (K_t, \bar{V}_t(\Omega_t), \rho_t)$

We see, then, that there are four fundamental information classes, and each brings into play specific algorithmic strategies. All the approaches assume that we are starting with a basic myopic model, and that we have algorithms for solving these problems. It is possible in theory to combine rolling horizon procedures and value functions, but value functions do offer a competing strategy for using forecasts of exogenous information processes to produce a forecast of a value of a resource in the future. Proximal point terms which incorporate expert knowledge can complement any model.

We can put all four information classes into a single decision function. Assume that we wish to manage equipment to move freight, including freight that is in the system now as well as forecasted freight within a planning horizon \mathcal{T}^{ph} . We could formulate a decision function that looks like

$$X_t^\pi(I_t) = \arg \max_{x_t \in \mathcal{X}_t} \sum_{t'=t}^{T^{ph}} \sum_{t''=t'}^{T^{ph}} \left(c_{t',t''} x_{t',t''} + \sum_{t'' > T^{ph}} \bar{V}_{t,t''}(R_{t,t''}^x(x_t)) - H(\rho(x), \rho) \right).$$

In this formulation, we are planning the flows of equipment within a planning horizon, using a point forecast of future information within this horizon. Equipment that becomes actionable past the end of the planning horizon is captured by a value function approximation, which we have represented as being separable over time periods. More commonly, optimizing over a planning horizon using a point forecast is done specifically to avoid the need for a value function. If value functions are used, these can be particularly nice mechanisms for handling stochastic forecasts. For this reason, we prefer to write our general purpose decision function as

$$X_t^\pi(R_t) = \arg \max_{x_t \in \mathcal{X}} \sum_{t'=t}^{T^{ph}} \left(c_{t,t'} x_{t,t'} + \sum_{t' > T^{ph}} \bar{V}_{t,t'}(R_{t,t'}^x(x_t)) - H(\rho(x), \rho) \right). \quad (61)$$

The idea of manipulating the information set available to a decision function is illustrated in Wu et al. (2003) in the context of the military airlift problem. This problem, simply stated, involves assigning cargo aircraft to move “requests” which are loads of freight and passengers. The

air mobility command currently uses a simulation package which captures a host of engineering details, but which uses very simple rules for choosing which aircraft should move each request. We use a version of these rules to represent the base policy. These rules work roughly as follows. The program maintains a list of aircraft in the order they are available, and requests to be moved, also in the order that they are available. Starting with the request at the top of the list, the program then determines whether the first available aircraft is eligible to move the request (there are a number of engineering restrictions). If not, it moves to the next aircraft on the list. As requests are satisfied, they are dropped from the list, and as aircraft are moved, they become available again in the future. Since a decision of which aircraft to choose does not compare different aircraft to find the best one, we can view this as a decision with very low information content. We refer to this as “rule-based, one aircraft, one request.”

The next step up is to choose a request, and then find the best out of a list of aircraft using a cost function. The list of aircraft that are considered include only those which are known now and actionable now (that is, we are limited to aircraft in the vector R_{tt}). This strategy is a cost-based, myopic policy, using a single aircraft and a list of requests. Next is a strategy that considers a list of requests and a list of aircraft, which requires solving an assignment problem to determine the best assignment of each aircraft to each request. This strategy is also restricted to aircraft and requests that are known now, actionable now. The fourth information set is the same as the third, but now we are allowed to consider aircraft and requests that are known now but actionable in the future. The fifth information set includes value functions, which are used to capture the cost of putting an aircraft into an airbase where it might break down, incurring maintenance costs that can vary significantly between airbases.

The results of these simulations are shown in figure 5, demonstrating steady improvement in costs as information is added to the decision function. We note that the value of increasing the information set depends on the problem. In addition, each increase in the information set adds to the computational requirements of the simulation. The use of value functions, for example, requires running the simulation iteratively, although it is often possible to estimate the value functions once and store these.

We did not include expert knowledge in this set of experiments since this information class exists only because the cost function is incomplete in some way. Imposing exogenous patterns through a proximal point term will always increase the costs. The appropriate measure here is the degree

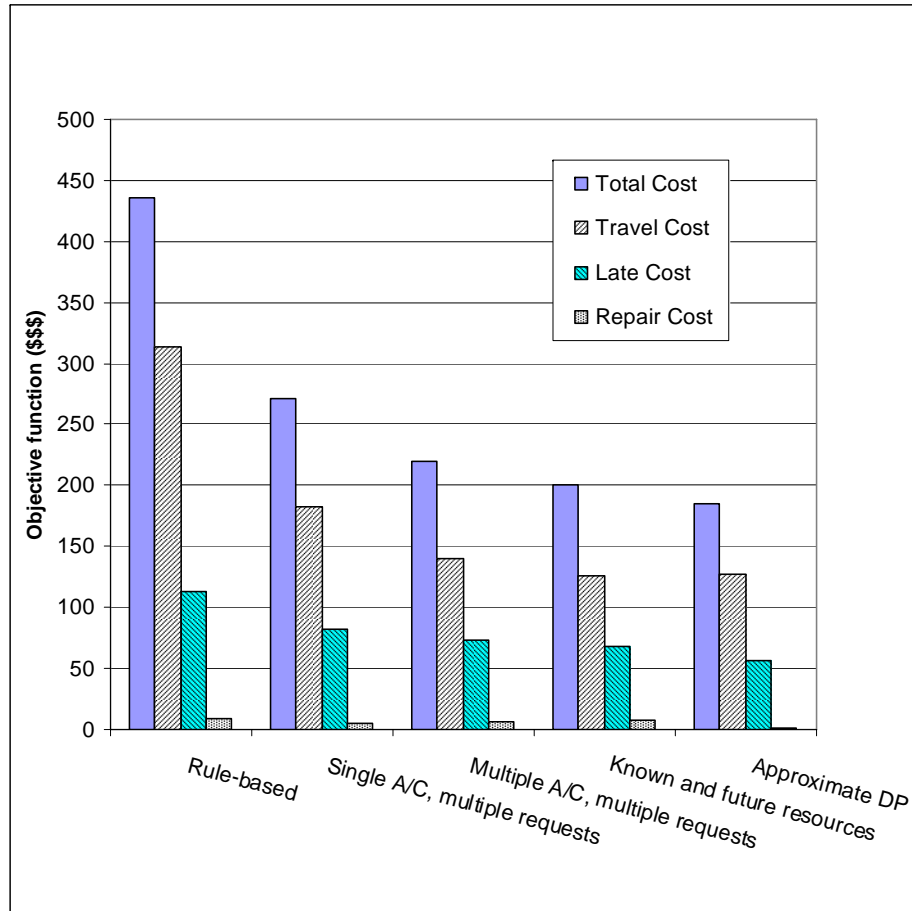


Figure 5: Total costs, broken down between travel costs, late penalty costs and repair costs, using four policies: 1) rule-based, chooses a single aircraft and single request and implements if feasible, 2) cost-based, considers a single aircraft and assigns to the best of a list of requests based on cost, 3) minimizes the costs of assigning a set of aircraft to a set of requests, limited to those that are both known and actionable now, 4) same as (3), but now includes aircraft and requests that are known now but actionable in the future, and 5) includes value functions to guide the choice of aircraft (from Wu et al. (2003)).

to which we are matching an exogenous pattern. To illustrate this process, we imposed a single exogenous pattern controlling the fraction of time that a particular aircraft type moved through a particular set of airbases. Without an exogenous pattern, this activity occurred approximately 20 percent of the time. Figure 6 shows the percentage of time this activity happened in the model, as a function of the weighting parameter θ . As θ is increased, the model produces results that are closer to the desired pattern, although there is a limit to this. In practice, if an expert says that we can do something 10 percent of the time, it is very likely that a solution that does it 15 or 20 percent of the time would be acceptable.

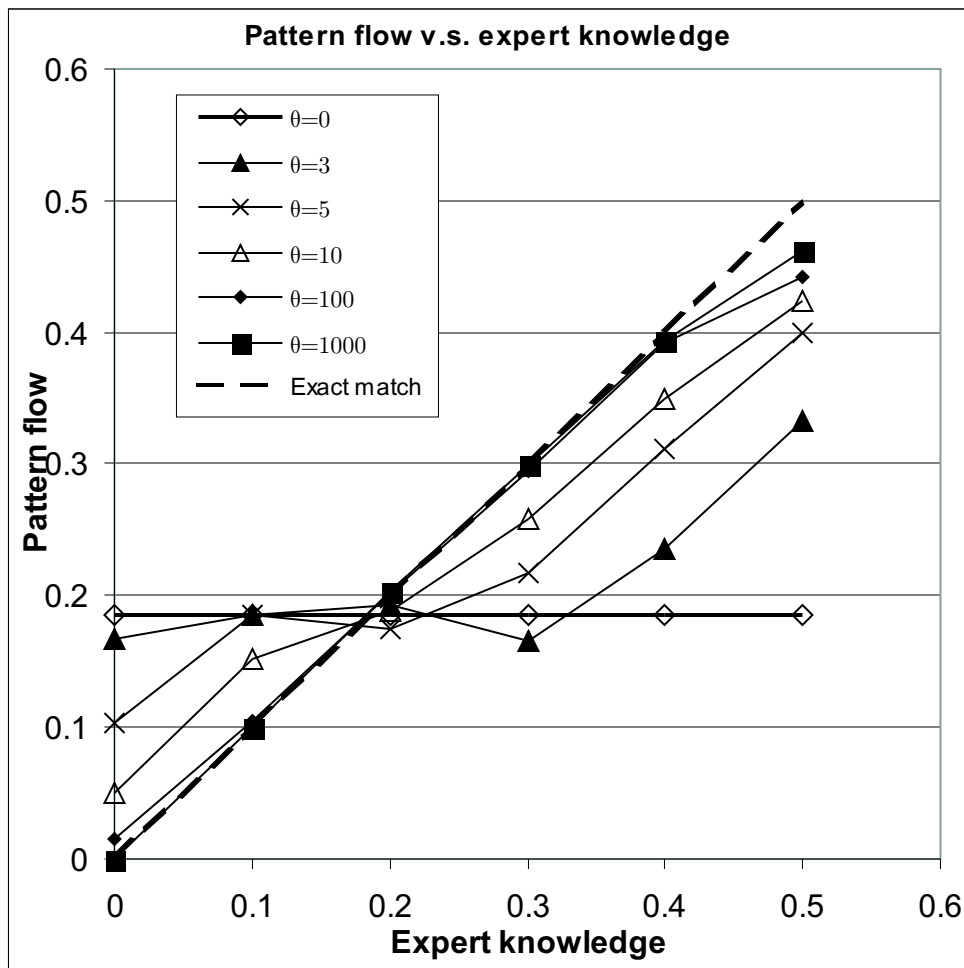


Figure 6: Actual versus desired flow of a particular aircraft through a particular airbase, as a function of the pattern weight parameter θ .

We are not trying to make the argument that all four information classes must be used in all models, but the exclusion of any information class should be recognized as such, and our experience has been that each will add value, especially when working on complex problems.

8.6 Bibliographic notes

There is a vast literature on myopic models and rolling horizon procedures that use point forecasts (virtually any deterministic dynamic model falls in this class). A number of authors have developed algorithms for exploiting the dynamic structure of these models (see, for example, Aronson & Chen (1986), Aronson & Thompson (1984)) although this work has been largely replaced by modern linear programming techniques. Stochastic programming models and algorithms are reviewed in

Infanger (1994), Kall & Wallace (1994), Birge & Louveaux (1997), Sen & Hige (1999). There are numerous texts on dynamic programming. Puterman (1994) is the best overview of classical methods of Markov decision processes, but covers only discrete dynamic programs, with algorithms that work only for very low dimensional problems. The use of the post-decision state variable has received very little attention. It is used implicitly in an exercise in Bellman's original text (Bellman (1957)), but not in a way that identifies its importance as a computational device. Godfrey & Powell (2002) use the post-decision state variable explicitly in the development of an approximate dynamic programming algorithm for fleet management, extending an approximation method used in a series of papers (Powell (1987), Cheung & Powell (1996)) where the technique was used implicitly. Van Roy (2001) introduces pre- and post-decision state variables, taking advantage of the fact that the post-decision state variable avoids the expectation in the determination of an optimal action. This property is not mentioned in any of the standard textbooks on exact or approximate dynamic programming (Bertsekas & Tsitsiklis (1996), Sutton & Barto (1998)), but it appears to be a powerful technique for high dimensional resource allocation problems. The explicit introduction of four classes of information appears to have been first done in Powell et al. (2001). The investigation of low-dimensional patterns is investigated in Marar et al. (to appear) and Marar & Powell (2004), and is applied to the military airlift problem in Powell et al. (2004).

9 Approximating value functions in dynamic programming

Section 8.3 introduces the idea of using approximate value functions, but did not fill in any of the details of how to do this. Approximate dynamic programming is an emerging field that must be proven experimentally on different problem classes, just as different heuristics have to be tested and proven on different problems. This section, which is easily skipped on a first reading, provides an introduction to the current state of the art in approximate dynamic programming, focused on the types of problems that arise in transportation and logistics.

Section 9.1 focuses on a problem where there is a single resource which might be characterized by a vector of attributes. This treatment lays the foundation for solving problems with many resources. Section 9.2 describes strategies for estimating continuous value function approximations, which are a central strategy for avoiding "curse of dimensionality" problems in dynamic programs.

9.1 Discrete value function approximations for a single resource

Fundamental to the challenge of allocating multiple resources is the problem of allocating a single resource. This helps us illustrate some basic strategies, but also highlights some issues that arise in more complex problems.

Our discussion is helped by a simple example. Assume we have the case of a nomadic trucker who moves around the country from state to state (within the United States). His attribute vector (for the moment) is the “state” that he is in (such as New York). From a state, he has to choose from a number of loads (requests to move freight) that he learns about after his arrival. The load he chooses should reflect the profit he expects from the load, plus the value of being in the state that the load terminates in (this will maximize our trucker’s long term profits). If we wanted to make our problem more realistic, we would include attributes such as the driver’s home domicile, how many days he has been away from his home, and the maintenance status of his truck (was there an equipment failure during his last trip that would require a repair at the destination?).

We let $a_{t-1} \in \mathcal{A}$ be the attribute of our driver at time $t - 1$ and $V_{t-1}(a_{t-1})$ be the expected future value of a driver that has attribute a at time $t - 1$. We can formulate this problem in two ways: as we have above using R_t as the state variable and x_t as our decision, or using a_t as the state variable and d as the decision where the set of potential decisions \mathcal{D}_t depends on the information available at time t . Recalling that $a^M(t, a_{t-1}, d)$ is the attribute vector produced by applying the decision d to a resource with attribute a_{t-1} , we can write Bellman’s equations as

$$V_{t-1}(a_{t-1}) = E\{\max_{d \in \mathcal{D}_t} (c_{t, a_{t-1}, d} + V_t(a^M(t, a_{t-1}, d))) | a_{t-1}\}. \quad (62)$$

Equation (62) is the more natural way to formulate this specific problem, but it disguises the relationship to problems with multiple resources. A natural way to bring the two formulations together is as follows. Since $\sum_{a \in \mathcal{A}} R_{ta} = 1$, we can write $V_t(R_t) = V_t(a_t) = \sum_{a \in \mathcal{A}} R_{ta} V_t(a)$ where $R_{ta_t} = 1$. Thus, for the single resource problem, $V_t(R_t)$ is linear in R_t .

If the attribute space \mathcal{A} is not too large, we can solve equation (62) using classical backward dynamic programming techniques. Not surprisingly, these problems do not generally exist in practice (the problems with resources with simple attribute vectors, such as containers or freight cars, are exactly the problems with many resources). The more interesting problems involve managing people (which tend to exhibit complex attribute vectors) or in multistop vehicle routing problems,

where the attribute vector has to capture the stops that a vehicle has already made. For these problems, we resort to our approximation strategy, where it is fairly obvious that a linear approximation is the appropriate form. This gives us

$$X_t^\pi(R_{t-1}^x, \omega^n) = \max_{x_t \in \mathcal{X}_t(\omega^n)} \left(c_t x_t + \sum_{a' \in \mathcal{A}} \bar{v}_{ta'}^n R_{ta'}^x(x_t) \right) \quad (63)$$

where $R_{ta'}^x(x_t) = \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}_a} x_{t,a,d} \delta_{a'}(t, a, d)$ is our post-decision resource vector. Our decision function X_t^π is written as depending on R_{t-1}^x and ω , but it actually depends on $R_t = R_{t-1}^x + \hat{R}_t(\omega)$.

The result of solving (63) is a random estimate \hat{v}_{ta}^n of the value of a resource with attribute a_{t-1} with the information available at time t . If we were using the simple one-pass algorithm, we could now update \bar{v}_{t-1}^{n-1} using

$$\bar{v}_{t-1,a}^n = (1 - \alpha^n) \bar{v}_{t-1,a}^{n-1} + \alpha^n \hat{v}_{ta}^n. \quad (64)$$

Again we note that \hat{v}_{ta}^n is indexed by t because it contains information from time t , while $\bar{v}_{t-1,a}^n$ represents an approximation of an expectation over all possible outcomes of W_t , and therefore implicitly contains only information up to time $t - 1$.

If we were to use the two-pass version of the algorithm, we would simulate the trajectory of our nomadic trucker using the policy $(X_{t'}^\pi)_{t' \geq t}$. We would then compute \hat{v}_{ta}^n as the cost of the trajectory over the interval (t, T) , and then update \bar{v} as we did in equation (64).

These procedures are known in the approximate dynamic programming literature as temporal-differencing methods (Sutton (1988), Sutton & Barto (1998), Bertsekas & Tsitsiklis (1996)) where they are normally described in the context of steady state problems. They can also be viewed as examples of stochastic gradient algorithms. To see this, we can write the problem of estimating the exact function $V_t(a)$ as one of solving

$$\min_{v_{ta}} f(v) = E \left\{ \frac{1}{2} (v_{ta} - \hat{V}_{ta})^2 \right\}. \quad (65)$$

where \hat{V}_{ta} is a random variable representing a measurement of the function with noise. We note that \hat{V}_{ta} is an unbiased estimate only when we use the two-pass version of the algorithm. Equation (65) can be solved by dropping the expectation, taking a Monte Carlo sample and then applying a

standard gradient algorithm

$$\begin{aligned}
v_{ta}^n &= v_{ta}^{n-1} - \alpha^n \nabla_v f(v_{ta}^{n-1}, \omega^n) \\
&= v_{ta}^{n-1} - \alpha^n (v_{ta}^{n-1} - \hat{V}_{ta}^n(\omega)) \\
&= (1 - \alpha^n) v_{ta}^{n-1} + \alpha^n \hat{V}_{ta}^n(\omega).
\end{aligned} \tag{66}$$

The procedure (66) is known as a stochastic approximation algorithm (Robbins & Monro (1951); see Kushner & Yin (1997) for a modern treatment of the field). In addition to some technical requirements on the properties of the function being estimated, the proof requires that $\hat{V}_{ta}^n(\omega)$ be an unbiased estimate of V_{ta} in the limit (biases are allowed initially as long as they go to zero reasonably quickly), and two conditions on the stepsizes: $\sum_{n=1}^{\infty} \alpha^n = \infty$ and $\sum_{n=1}^{\infty} (\alpha^n)^2 < \infty$. These requirements basically require that the stepsize decrease to zero arithmetically, as in $\alpha^n = 1/n$.

This simple problem highlights an issue that plagues all approximate dynamic programming methods. Assume that we start with an initial approximation $\bar{v}_{ta} = 0$ for all t and a . In the case of our nomadic trucker, this means that initially, he will prefer long loads with high initial rewards over shorter loads with low initial reward. In fact, a shorter load might be more profitable in the long run if it takes him to a region with high profit loads. If we only update the values of states that he visits, then he will tend to only visit states that he has already visited before.

One way to avoid this behavior is to begin by initializing \bar{v}_t with an upper bound. This way, if our trucker turns down a load to a state that he has never visited, then we can be assured that we are not turning away an option that might prove to be quite good. In practice, this strategy works poorly, because it is more likely that he will always choose the load that goes to a state that he has never visited (for our problem, it is not just a state, but also a state at a point in time). If the attribute vector includes other dimensions, then our attribute space \mathcal{A} can grow very quickly (a nice example of the curse of dimensionality).

The problem of visiting states in order to estimate the value of being in the state is called the *exploration* problem in dynamic programming. There is no cure for this problem, but there are strategies that help overcome it. One is to use “off-policy iterations” which means to choose actions that are not optimal given the current approximations, but forces the system to visit new states. In real problems, the number of states and actions tends to make such strategies impractical

(at a minimum, they have very slow convergence). A more effective strategy is to exploit problem structure. For example, assume that one attribute of our truck driver is the number of days away he has been from home, and suppose that we are able to establish that the value of the driver declines monotonically with this quantity. We can then use this structural property to help estimate the values of states we might never have visited.

The problem of estimating high-dimensional functions has open theoretical and practical questions. From a theoretical perspective, existing proofs of convergence for discrete value functions requires visiting all states infinitely often. This generally requires an algorithm that combines optimizing iterations (where we choose an action based on our approximate policy) with learning iterations (where we might choose actions at random). From a practical perspective, learning steps are only of value with low-dimensional action spaces. Practical considerations tend to focus on the rate of convergence, which tend to be highly problem dependent.

9.2 Continuous value function approximations

A popular strategy for dealing with the curse of dimensionality has been to replace the value function with a continuous approximation (Bellman & Dreyfus (1959)). Particularly appealing are the use of low-order polynomials. In this section, we describe several strategies for estimating continuous approximations.

9.2.1 Basis functions

A strategy for estimating a continuous approximation of a value function is to use the concept of basis functions. Assume, for example, that we are managing a fleet of containers of different types, and we believe that the number of containers of each type at each location is an important determinant of the value of a particular allocation. We could devise a function, call it $\phi_a(R) = \sqrt{R_{ta}}$. $\phi_a(R)$ takes the square root of this quantity because we believe that there are diminishing returns from additional capacity. In this case, we would call the square root of the number of containers with attribute a “a feature” of our problem. Another feature might be the total number of containers of all types at a location i . We might then create a new class of functions $\phi_i(R) = \sqrt{\sum_{a \in \mathcal{A}_i} R_{ta}}$ where \mathcal{A}_i is the set of attribute vectors that correspond to resources at a particular location i . Another feature could be the number of containers minus the forecasted number of orders. It is up to our imagination and insight to identify what appear to be important features. Let \mathcal{F} be the set of

all these functions, known as *basis functions* or *features*. We can formulate an approximate value function as a linear combination of our basis functions

$$\bar{V}(s|\theta) = \sum_{f \in \mathcal{F}} \theta_f \phi_f(s). \quad (67)$$

Now the challenge is to determine the appropriate weights $(\theta_f)_{f \in \mathcal{F}}$. We illustrate two strategies for doing this. The first is to simply apply a stochastic gradient algorithm. We generalize slightly the minimization problem in (65) as follows

$$\min_{\theta} E \left\{ \frac{1}{2} (\bar{V}^{n-1}(s|\theta) - \hat{V}(s))^2 \right\}.$$

Solving this using a stochastic gradient algorithm produces updates of the form

$$\theta^n = \theta^{n-1} - \alpha^n ((\bar{V}^{n-1}(s|\theta) - \hat{V}(s, \omega)) \nabla_{\theta} \bar{V}(s, \omega | \theta)). \quad (68)$$

The challenge that this class of update procedures faces is the problem of scaling the stepsize, since the units of θ and the units of the updating term will be completely different. An alternative strategy that avoids this problem takes a sample $\hat{\Omega}$ and then finds θ that minimizes the deviation between the approximation and the sample

$$\theta^n = \arg \min_{\theta} \frac{1}{|\hat{\Omega}|} \sum_{\omega \in \hat{\Omega}} \left(\frac{1}{2} (\bar{V}^{n-1}(s, \omega | \theta) - \hat{V}(s, \omega))^2 \right). \quad (69)$$

We can then use the value of θ that solves (69), or smooth this estimate with the previous estimate.

An introduction to the use of basis functions can be found in Bertsekas & Tsitsiklis (1996), with an in-depth treatment of the theoretical foundation in Tsitsiklis & Van Roy (1997). However, there is virtually no experimental work in the area of transportation and logistics. There are two practical challenges: identifying effective basis functions that capture the important properties of the problem, and the second is ensuring that the rate of convergence is sufficiently fast.

9.2.2 Auxiliary functions

Consider the case of a continuous resource allocation problem (where R_t is continuous). This means that we can find a stochastic gradient of $\tilde{V}_t^n(R_t(\omega^n))$ in equation (49). Let

$$\hat{v}_t^n = \nabla_{R_t} \tilde{V}_t^n(R_t(\omega^n))$$

be a stochastic gradient of the function. Now let $\bar{V}_t^0(R_t)$ be a conveniently chosen continuous approximation (perhaps a low order polynomial). We can improve our approximation using the updating equation

$$\bar{V}_t^n(R_t) = \bar{V}_t^{n-1}(R_t) + \alpha^n (\hat{v}_t^n - \nabla_{R_t} \bar{V}_t^{n-1}(R_t)) R_t. \quad (70)$$

This is the primary step of the SHAPE algorithm (Cheung & Powell (2000)). It starts with an arbitrary (but presumably carefully chosen) initial approximation, and then tilts it using stochastic gradient information. The second term of equation (70) is a linear updating term that adds the difference between the stochastic gradient of the real function and the exact gradient of the approximate function. Each updated function has the same basic form as the initial approximation. This can be a nice simplification if it is necessary to precompute derivatives.

The SHAPE algorithm is optimal for two-stage, continuously differentiable problems. For non-differentiable problems, and for multistage problems, it is an approximation. In principle the initial approximation could be estimated from a set of basis functions, allowing both methods to be used as a hybrid strategy.

Computational testing of these methods is limited. Their success appears to be highly dependent on a good choice of initial approximation. Simply choosing low-order polynomials because they are convenient is not likely to be successful.

9.2.3 Linear functional approximations

We have already seen in the case of managing a single resource that the value function can be written as a linear function of the resource vector. Linear approximations can also be effective when we are managing very large numbers of resources. For example, shipping companies might manage fleets of hundreds of thousands of containers. Our ability to effectively manage resources depends primarily on our ability to estimate the slope of the function, rather than the function itself. When there are large numbers involved, the slopes tend to be fairly stable.

Throughout this section, we assume that \hat{v}_{ta} is a stochastic gradient of either $\tilde{V}_t(R_t)$ with respect to R_{ta} (equation (49)), or of $V_t^\pi(R_t, \omega)$ (equation (52), computed using a backward pass). The choice of which version to use is primarily one of trading off speed of convergence with algorithmic complexity (backward passes can be hard to implement but can dramatically improve

results). Given an estimate of the slope, we would then smooth on the slope (as in equation (64)) to obtain an estimate of the slope, producing the following approximation

$$\tilde{V}_t^n (R_t^n(\omega^n)) = \max_{x_t \in \mathcal{X}_t(\omega^n)} \left(c_t x_t + \sum_{a \in \mathcal{A}} \bar{v}_{ta}^{n-1} R_{t,a}^x(x_t) \right). \quad (71)$$

Using our $\delta(\cdot)$ function, we can write equation (71) as

$$\tilde{V}_t^n (R_t^n(\omega^n)) = \max_{x_t \in \mathcal{X}_t(\omega^n)} \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}_a} c_{tad} x_{tad} + \sum_{a' \in \mathcal{A}} \bar{v}_{ta'}^{n-1} \left(\sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}_a} \delta_{t,a'}(t, a, d) x_{tad} \right) \quad (72)$$

$$= \max_{x_t \in \mathcal{X}_t(\omega^n)} \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}_a} c_{tad} x_{tad} + \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}_a} \bar{v}_{t,a^M}^{n-1}(t, a, d) x_{tad} \quad (73)$$

$$= \max_{x_t \in \mathcal{X}_t(\omega^n)} \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}_a} c_{tad} x_{tad} + \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}_a} \bar{v}_{t,a^M}^{n-1}(t, a, d) x_{tad} \quad (74)$$

$$= \max_{x_t \in \mathcal{X}_t(\omega^n)} \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}_a} (c_{tad} + \bar{v}_{t,a^M}^{n-1}(t, a, d)) x_{tad}. \quad (74)$$

We obtain equation (72) by using $\delta_{t,a'}(t, a, d) = 1$ for $a' = a^M(t, a, d)$ and 0 otherwise, producing the more compact expression in (73). Equation (74) is obtained by simply rearranging terms, which shows that we are now solving a problem with the same structural form as the original myopic problem, with a modified set of costs.

This structure is especially important in transportation problems since the original myopic problem might be fairly difficult. If we are lucky, it is an assignment problem, transportation problem or some linear program. But in some settings it is a set partitioning problem, vehicle routing problem, or integer multicommodity flow problem. For these problems, it is especially helpful when the value function approximation does not complicate the problem further.

Not surprisingly, linear value functions do not always work well. It is very hard to tune a linear function to attract just the right number of resources. Nonlinear functions are simply more stable, but as we show below, they introduce considerable complexity.

9.2.4 Separable piecewise-linear functional approximations

When we are managing discrete resources (freight cars, trailers, people, aircraft, locomotives) it is especially important to obtain integer solutions. This is simplified considerably when we use separable, piecewise-linear approximations which are linear for noninteger values of the number of

resources. For this strategy, we write our approximation as

$$\bar{V}_t(R_t) = \sum_{a \in \mathcal{A}} \bar{V}_{ta}(R_{ta})$$

where

$$\bar{V}_{ta}(R_{ta}) = \sum_{r=0}^{\lfloor R_{ta} \rfloor - 1} \bar{v}_{ta}(r) + \bar{v}_{ta}(\lfloor R_{ta} \rfloor)(R_{ta} - \lfloor R_{ta} \rfloor). \quad (75)$$

When we introduce this approximation in equation (49) we obtain

$$\hat{V}_t^n(R_t^n(\omega^n)) = \max_{x_t \in \mathcal{X}_t(\omega^n)} \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}_a} c_{tad} x_{tad} + \sum_{a' \in \mathcal{A}} \bar{V}_{ta'}(R_{ta'}) \left(\sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}_a} \delta_{t,a'}(t, a, d) x_{tad} \right). \quad (76)$$

Assume that the original myopic problem is a pure network. We are interested in the conditions under which the introduction of value functions destroys this network structure. The constraint set $\mathcal{X}_t(\omega^n)$ typically contains upper bounds representing vehicle capacity constraints, network constraints, or simply constraints on the number of tasks to be served. Let u_{td} be the upper bound on the total number of resources acted on by a particular decision type

$$\sum_{a \in \mathcal{A}} x_{tad} \leq u_{td}. \quad (77)$$

Equation (77) has the effect of creating a bundle constraint. This does not pose a problem when we use linear value functions, as we demonstrated that any linear value function is equivalent to solving the original problem with a modified set of costs. But when we use nonlinear value functions, the network structure is destroyed.

Of particular interest are problems that satisfy the Markov property:

Definition 9.1 *A resource allocation problem satisfies the Markov property if*

$$a^M(t, a, d) = a^M(t, a', d) \quad \forall a' \in \mathcal{A}.$$

The Markov property means that the attributes of a resource after being acted on are independent of the attributes of the resource. A special class of problems that satisfy the Markov property are

known as single commodity flow problems. In this problem class, the attribute vector consists purely of the location of a resource. A task consists of an origin and a destination, so after completing the task, the attribute of the resource equals the destination of the task (and is therefore independent of the origin). Another example arises when we have to maintain an attribute of whether a container is clean or dirty. Assume there is freight that is classified as clean or dirty. Only a clean container can move clean freight, but any container can move dirty freight. However, if a clean car moves dirty freight, then it becomes dirty. Such a problem would also satisfy the Markov property.

When a problem possesses the Markov property, the attributes of the resource after being acted on are completely determined by the decision rather than the attributes of the resource. As a result, we can create a node for resources that have been acted on by a particular type of decision, and we do not have to track the identity of the origin of the resource after it completes a task (or any other action). As a result, it is possible to formulate the problem as a pure network.

When problems do not possess the Markov property, the resulting problem will not, in general, be a pure network. But the use of piecewise linear, separable value function approximations appears to produce integer multicommodity flow problems with very tight LP relaxations. Experimental evidence is that when these problems are solved as continuous linear programs, the solution returned is integer the vast majority of the time (99.9 percent).

9.3 Bibliographic notes

The field of approximate dynamic programming is relatively young. The first significant reference is Bertsekas & Tsitsiklis (1996) which gives a general introduction to a variety of methods, but without presenting specific algorithms. Sutton & Barto (1998) approach the topic from the perspective of reinforcement learning. Temporal-difference learning, a form of stochastic approximation procedure, was first introduced by Sutton (1988). The vast majority of the literature on dynamic programming appears to focus on discrete representations of the value function, but there has been a steady literature on the use of continuous approximations, beginning as early as Bellman & Dreyfus (1959). Tsitsiklis & Van Roy (1997) investigate in depth the convergence properties of temporal-difference methods using continuous approximations. The SHAPE algorithm was introduced by Cheung & Powell (2000). The theory behind the estimation of linear value function approximations is based on the seminal work on stochastic approximation methods by Robbins & Monro (1951) and Blum (1954) (see Kushner & Yin (1997) for a modern review of these techniques). The use of separable,

piecewise linear approximations was first introduced by Cheung & Powell (1996) who used a static approximation. Godfrey & Powell (2001) proposes an adaptive learning algorithm for piecewise linear, separable approximations in the context of two-stage resource allocation problems, which is extended in Godfrey & Powell (2002) for multistage, single commodity problems. Topaloglu & Powell (to appear) shows that these techniques also work quite well for multicommodity flow problems with a range of forms of substitution.

10 The organization of information and decisions

Another important issue in the modeling of decisions and information is how they are organized. Large transportation operations are typically characterized by different decision makers controlling different parts of the problem with different information. In this section, we provide some basic notation to model how organizations are controlled.

We adopt the convention that a decision-maker is referred to as an “agent” which is assumed to have sole control over a set of decisions. We let:

\mathcal{Q} = Set of decision-making agents.

\mathcal{D}_q = The set of decisions controlled by agent q .

Our first challenge is to formalize the decision sets \mathcal{D}_q . We assume that the sets $(\mathcal{D}_q)_{q \in \mathcal{Q}}$ are mutually exclusive and collectively exhaustive.

There are three dimensions to the set \mathcal{D}_q : the types of decisions an agent can make, the types of resources an agent can act on, and the time periods at which the decisions might be implemented. For example, a locomotive planner for a railroad might have the responsibility for planning the movements of locomotives on Monday, Tuesday, and Wednesday. He can only control locomotives that are in service; a local maintenance supervisor has control over locomotives that are classified as being “in maintenance” (it is common for a locomotive planner to call the local maintenance supervisor to determine when a locomotive that is in the shop will become available). Furthermore, he can make decisions about coupling and uncoupling locomotives to and from trains, but does not make decisions about buying, selling, leasing and storing locomotives. To capture this process, we let:

\mathcal{A}_q = Subset of the attribute space for subproblem q , where $\cup_{q \in \mathcal{Q}} \mathcal{A}_q = \mathcal{A}$ and $\mathcal{A}_{q_1} \cap \mathcal{A}_{q_2} = \phi$ when $q_1 \neq q_2$. Implicit in the definition of the attribute space is:

\mathcal{C}_q^D = Set of control classes associated with subproblem q . As a rule, a subproblem is formulated for a specific type of decision, so the set \mathcal{C}_q^D is implicit in the formulation of the problem.

\mathcal{D}_a^c = Set of decisions in control class c that can be applied to a resource with attribute vector a .

\mathcal{T}_q^{ih} = The *implementation horizon* for subproblem q . This is the set of time periods during which subproblem q controls the decisions. Since time is a dimension of the attribute vector, we may state that $a \in \mathcal{A}_q \Rightarrow a_{actionable} \in \mathcal{T}_q^{ih}$.

It is important to emphasize that the subsets \mathcal{A}_q do not necessarily (or even typically) imply a geographical decomposition, although this certainly can be the case. At a railroad, locomotives are managed regionally (with some network-wide coordination); freight cars are managed network-wide, but are decomposed by freight car type.

Our decision set for agent q is now given by:

$$\begin{aligned} \mathcal{D}_q &= \text{Subset of decisions in subproblem } q. \\ &= \{d \in \mathcal{D}_a^c, c \in \mathcal{C}_q^D, a \in \mathcal{A}_q, a_{actionable} \in \mathcal{T}_q^{ih}\} \end{aligned}$$

The definition of \mathcal{D}_q includes the time periods for which the decisions may be implemented, which produces a more compact notation. In some cases, the indexing of when a decision is to be implemented needs to be made explicit, so we can use (t, q) to capture the combined information.

Before, the information available to make a decision was represented implicitly either by indexing time t , or explicitly through a state variable such as S_t or R_t . When we model multiagent systems, we have to be more explicit. For this reason, we define:

I_{tq} = The information content of subproblem q at time t .

It is best to think of I_{tq} as a set of functions that return data that is needed to make a decision: the amount of equipment, people, and freight, available now and in the future (recall that the index t here refers to information content, not actionability).

Another dimension of multiagent systems is capturing the property that decisions by agent q can have an impact on another agent q' (for example, sending a piece of equipment from one region to the next, or when the person who buys and sells equipment changes the amount of equipment that a planner can work with). We capture these interactions by defining the following:

Definition 10.1 *The forward reachable set $\vec{\mathcal{I}}_q$ of subproblem q is the set of subproblems q' with resource states $a' \in \mathcal{A}_{q'}$ that can be reached by implementing a single decision $d \in \mathcal{D}$ on at least one state $a \in \mathcal{A}_q$. More precisely, the forward reachable set of subproblem q is*

$$\vec{\mathcal{I}}_q = \{q' \in \mathcal{Q} \setminus q \mid \exists a \in \mathcal{A}_q, d \in \mathcal{D}_q \text{ where } M(a, d, \cdot) \mapsto (a', \cdot, \cdot), a' \in \mathcal{A}_{q'}\}. \quad (78)$$

For completeness, we also define:

Definition 10.2 *The backward reachable set $\overleftarrow{\mathcal{I}}_q$ of subproblem q is the set of all subproblems for which subproblem q is forward reachable. More precisely, the backward reachable set of subproblem q is*

$$\overleftarrow{\mathcal{I}}_q = \{q' \in \mathcal{Q} \setminus q \mid q \in \vec{\mathcal{I}}_{q'}\}. \quad (79)$$

This allows us to define decision vectors for each agent:

$$\begin{aligned} x_{taa'} &= \sum_{d \in \mathcal{D}_q} x_{tad} \delta_{a'}(t, a, d), \\ x_{tqa'} &= \{x_{taa'}, a \in \mathcal{A}_q\}, \\ x_{tqq'} &= \{x_{tqa'}, a' \in \mathcal{A}_{q'}\}, \\ x_{tq} &= \{x_{tqq'}, q' \in \vec{\mathcal{I}}_q\}. \end{aligned}$$

We adapt our notation for making decisions by defining:

$$\begin{aligned} X_{tq}^\pi(I_{tq}) &= \text{A function that determines } x_{tad} \text{ for } a \in \mathcal{A}_q, d \in \mathcal{D}_q \text{ using the information available} \\ &\text{in } I_{tq}, \end{aligned}$$

where I_{tq} is the information known by agent q at time t . As we did before, we can create a general

purpose decision function that uses all four information classes

$$X_{tq}^\pi(I_{tq}) = \arg \max_{x_{tq}} c_{tq} x_{tq} + \sum_{q' \in \vec{\mathcal{I}}_q} \bar{V}_{tqq'}(R_{tq'}(x_{tqq'})) - H(\rho(x_{tq}), \rho_q). \quad (80)$$

Shapiro & Powell (to appear) describe this strategy in depth and give a method for estimating the value functions $\bar{V}_{tqq'}(R_{tq'}(x_{tqq'}))$.

11 Illustrative models

We illustrate these ideas in the context of three problem areas: rail car distribution, load matching for truckload motor carriers, and batch processes that arise in less-than-truckload motor carriers. The car distribution problem is characterized by a low dimensional attribute space (it is roughly a multicommodity flow problem) with an interesting set of dynamic information processes. The load matching problem introduces the challenge of modeling people (characterized by high dimensional attribute spaces). Also, we demonstrate the modeling of a two-layer resource allocation problem (drivers and loads). Despite their differences, both of these problems have the flavor of time-staged linear programs. The batch processes that arise in less-than-truckload motor carrier illustrates how we can handle the types of fixed-charge problems that arise in network design.

11.1 Dynamic flow problems for rail car distribution

Our car distribution problem provides an opportunity to illustrate the notation that we have introduced. Section 11.1.1 provides a model of an actual car distribution system developed for a major railroad. Then, section 11.1.2 illustrates several optimization models that are used in practice. Section 11.1.3 summarizes the results of some experiments run with an actual dataset from a major railroad. Section 11.1.4 closes with some observations about this application.

11.1.1 A dynamic model

The car distribution problem basically involves assigning cars to orders. Cars can be moved empty or assigned to orders and then moved loaded. Cars or orders that are not acted on are held. Orders that are not satisfied within a reasonable time (typically within the same week) are considered lost, but in practice some orders might simply be held to the next week. When a car becomes empty at

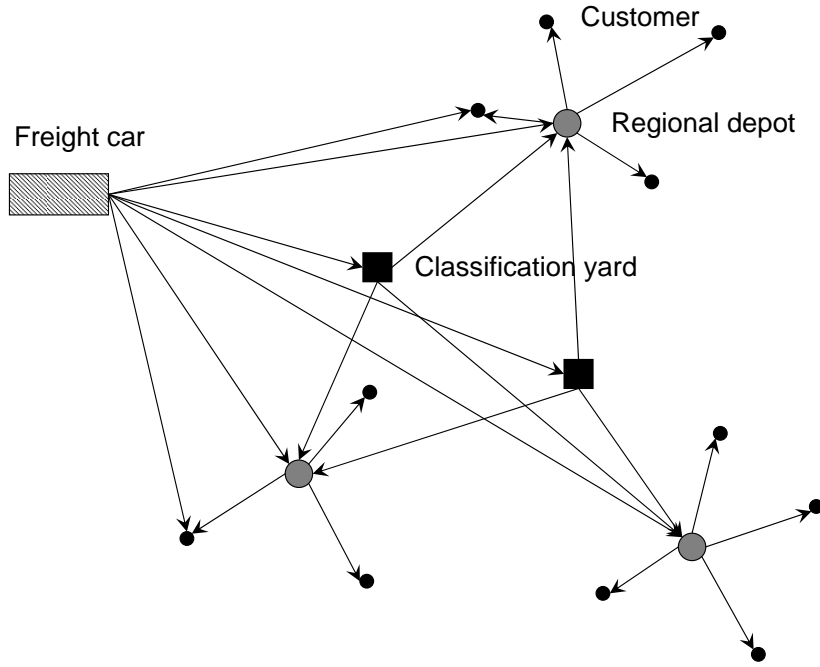


Figure 7: Car distribution through classification yards

a shipper, the railroad has roughly three options which are illustrated in figure 7: assign the car to a pending order, move the car to a “regional depot” from which it can be used to satisfy orders from other shippers in the region, or send the car to a “classification yard.” A classification yard is a major sorting facility, from which cars might be sent to anywhere in the network.

Our problem, then, involves two “resource classes,” namely cars and orders. We model cars using:

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

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

$R_{t,t'}^c =$ 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.

$$R_{t,t'}^c = (R_{t,t',a}^c)_{a \in \mathcal{A}}$$

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

For our problem, we model the information process (the arrival of information and the making of decisions) in 24-hour increments. By contrast, we model the physical movement of resources in

continuous time. As a result, our attribute vector consists of

$$a = \begin{bmatrix} a_{location} \\ a_{car_type} \\ a_{actionable} \end{bmatrix}$$

The element of $a_{actionable}$ gives the time, down to the minute, of when a car is able to move. Note that there are some important modeling decisions in how the attribute is handled. Assume, for example, that a car at origin o has to pass through classification yards (where cars are sorted and can be rerouted) at i and j on its path to destination d . If the car is sent to d with expected arrival time t' , we can update its attribute vector to include $a_{location} = d$ and $a_{actionable} = t'$. This means that we will not consider rerouting the car at i or j . On the other hand, we might wish to consider rerouting, but we do not want to ignore that it is already on a path to d . The railroad generates a piece of paper called a work order that routes the car to d ; if we reroute the car to a different destination, this paperwork has to be changed. If we wish to allow rerouting, but without forgetting the original destination of a car, we would simply add a new dimension to a which we might call $a_{destination}$ which would be the ultimate destination of the car, and we would interpret $a_{location}$ as the next location where a car is able to be rerouted (so, as the car departs from o heading to i , we would set $a_{location} = i$ and $a_{destination} = d$).

Orders are modeled in a similar way using:

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).

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

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

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

The attributes of orders consist of

$$b = \begin{bmatrix} b_{origin} \\ b_{destination} \\ b_{car_type} \\ b_{actionable} \end{bmatrix}$$

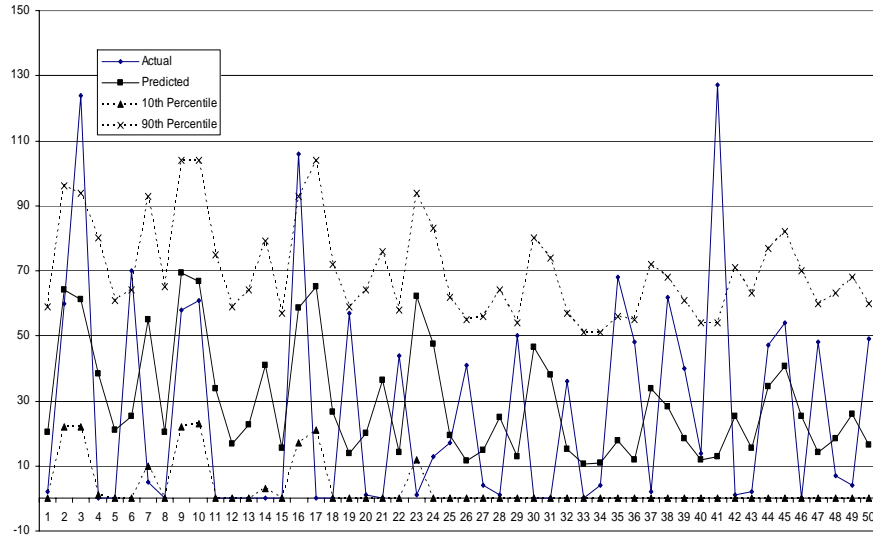


Figure 8: Actual vs. predicted forecasts of future demands, showing the 10th and 90th percentiles

Here b_{car_type} represents the preferred car type, but substitution opportunities exist. $b_{actionable}$ is the exact time that the order is available to be picked up. We note that when an order becomes known, it is not necessarily the case that all the attributes of an order become known at the same time. So we may adopt a convention that, say, $b_{destination} = '-'$ means that the destination is not yet known.

Our resource, vector, can now be written

$$R_t = (R_t^c, R_t^o).$$

One of the challenges of car distribution is dealing with a high level of uncertainty. Figure 8 illustrates the actual and predicted customer demand, with 10th and 90th percentiles. In addition, there are other forms of uncertainty: the process of loaded cars becoming empty, the transit times, and the acceptability of a car to a shipper (is it clean enough?). There are five types of information processes that railroads have to deal with:

- 1) Orders - Customers call in orders for cars, typically the week before when they are needed. The 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.

Railroads manage this uncertainty using several strategies. First, there is the fact that customers make their orders partially known in advance. For example, customers might book their orders the week before, although it is common for them to do so toward the end of the week. However, there can be discrepancies between what they book in the computer vs. what they really need (which might be communicated by phone). For example, if a railroad is running short on cars, customers have a tendency to overbook. Second, railroads depend on various forms of substitution when they cannot fill an order. Three forms of substitution come into play:

- 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.

It is common to model the decisions of moving cars from one location to another, but in practice car distribution is characterized by a number of decisions, including:

- 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 might 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. To represent these decisions, we define:

\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 customer location).

\mathcal{D}^o = The decision to assign a car to a type of order. For $d \in \mathcal{D}^o$, we let $b_d \in \mathcal{B}$ be the attributes of the order type associated with decision d .

\mathcal{D}^{rd} = The decision to send a car 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^ϕ = The decision to hold the car (“do nothing”).

The next step is to model the exogenous information processes. We start by modeling the arrivals of new resources, which includes both cars and orders. For example, a railroad might send a car “off line” to another railroad. When it returns to the railroad, it is as if a new car is arriving to the system. In some cases, a car sent to a shipper can be modelled as a car that is effectively disappearing from the network; when the car is released and returned from the shipper, it is a new arrival. New orders, of course, arrive in the form of phone calls or entries into a computer database (for example, through a web site).

We take advantage of our ability to separate the arrival process of information from the physical arrival of cars and orders. For example, we would represent new orders using:

$\hat{R}_{tt'}^o$ = The vector of changes to the order vector (new customer orders, changes in orders) that arrive during time interval t that become actionable at time $t' \geq t$.

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

We would define similar vectors $\hat{R}_{tt'}^c$ and \hat{R}_t^c for cars. Let W_t be our general variable representing the information arriving at time t , where

$$W_t = (\hat{R}_t^o, \hat{R}_t^c).$$

11.1.2 Optimization formulation

The most basic optimization model for car problems is the one that matches known cars to known orders over a predetermined horizon (even the most basic model captures the fact that some cars and orders are actionable in the future). This model can be stated as

$$\max_x \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} c_{0ad} x_{0ad} \quad (81)$$

subject to

$$\sum_{d \in \mathcal{D}^c} x_{0ad} = R_{0a}^c \quad a \in \mathcal{A} \quad (82)$$

$$\sum_{a \in \mathcal{A}} x_{0ad} \leq R_{0b_d}^o \quad d \in \mathcal{D}^o \quad (83)$$

$$x_{0ad} \in Z_+ \quad a \in \mathcal{A}, d \in \mathcal{D}^c. \quad (84)$$

Equation (82) captures flow conservation constraints on cars only, while equation (83) ensures that we do not assign more cars to orders than we have orders (effectively enforcing flow conservation on orders). This model follows the normal convention of modeling cars as an *active* resource (resources that we actively modify) while orders are a passive resource (they only change their status if they are assigned to a car). Recall that in equation (83), for every element of $d \in \mathcal{D}^o$ there is an order *type* with attribute b_d .

The next step up in sophistication is a model that incorporates point forecasts of cars and orders that become known in the future (this model is in use by at least one railroad). Let \bar{R}_t^c and \bar{R}_t^o be point forecasts of cars and orders of information that will become available during time period t . This model can be written as

$$\max_x \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} c_{0ad} x_{0ad} \quad (85)$$

subject to

$$\sum_{d \in \mathcal{D}} x_{0ad} = R_{0a}^c + \sum_{t \in \mathcal{T}^{ph} \setminus 0} \bar{R}_{ta}^c \quad d \in \mathcal{D}^c, a \in \mathcal{A} \quad (86)$$

$$\sum_{a \in \mathcal{A}} x_{0ad} \leq R_{0b_d}^o + \sum_{t \in \mathcal{T}^{ph} \setminus 0} \bar{R}_{tb_d}^o \quad d \in \mathcal{D}^o \quad (87)$$

$$x_{0ad} \in Z_+. \quad (88)$$

Note that this model aggregates forecasts of all future information about cars and orders. We retain the information about when the order will be actionable in the vector b_d , so that decisions to assign, say, a car available now to an order that is forecasted to arrive at time t' which will then be actionable at time t'' , can properly account for the difference between the available time of the car and the available time of the order.

Another strategy is to use our dynamic programming approximations as we outlined above. If we use a (possibly nonlinear), separable value function, we would find ourselves solving

$$X_t^{\pi,n}(R_t) = \arg \max_{x_t} \left(c_t x_t + \sum_{t' \geq t} \sum_{a \in \mathcal{A}} \bar{V}_{t,t'a}^{n-1}(R_{t,t'a}^x(x_t)) \right) \quad (89)$$

subject to

$$\sum_{d \in \mathcal{D}} x_{tad} = R_{t,ta}^c \quad a \in \mathcal{A} \quad (90)$$

$$\sum_{a \in \mathcal{A}} x_{tad} \leq R_{tb_d}^o \quad d \in \mathcal{D}^o \quad (91)$$

$$x_{tad} \in Z_+. \quad (92)$$

Equation (90) limits us to acting on cars that are actionable now, while equation (91) says that we cannot move more cars loaded than the orders we know about now. This problem is generally quite easy to solve, but does require the iterative learning logic described in section 9.

11.1.3 Some numerical experiments

The approximate dynamic programming algorithm was run on a dataset from a major railroad using nonlinear, separable value functions. The results are presented in figure 9 which shows total profits, along with total revenues, empty movement costs and service penalty costs. The figure shows a steady improvement in total profits as the objective function improves. The total revenue drops in the early iterations but rises back to approximately the same level as the first iteration, indicating that our improvement is through better repositioning rather than higher coverage. These experiments indicate that the adaptive learning logic works, but raises the question: exactly what is it doing that is smarter than a myopic model?

A common misconception is that we do not need to look into the future because customer orders are known in advance. For the rail dataset, customers orders are generally known the week before,

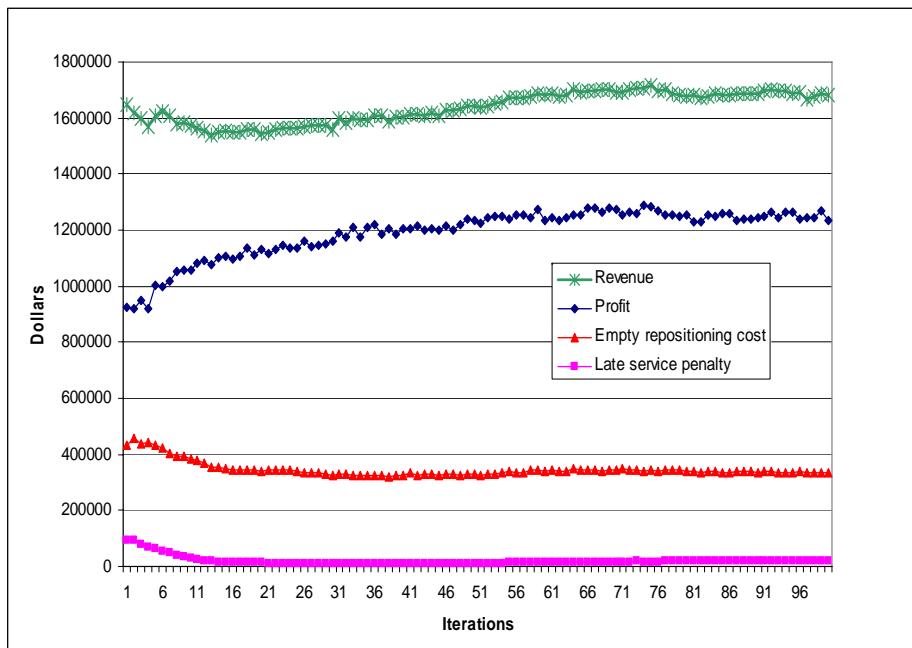


Figure 9: The contribution of adaptive learning to revenue, costs and overall profits.

Empty miles as a percent of total miles

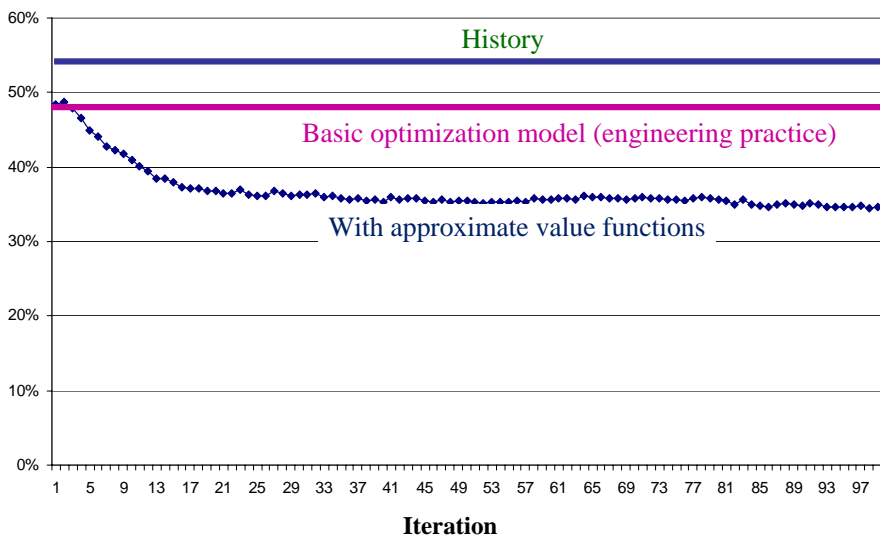


Figure 10: Empty miles as a percent of total a) in history, b) optimized using a myopic model, and c) optimized using an approximate dynamic programming procedure

with the bulk of customers calling in their orders on Wednesday and Thursday. Empty movement times are typically less than five days, which suggests that the prebook time allows the railroad to wait until orders are known before moving a car. Figure 10 shows the percent of total miles that are empty in history (approximately 54 percent for this dataset), from a myopic optimization

model (which uses information about cars and orders that become available in the future), and from an approximate dynamic programming model. The myopic model reduces empty miles to approximately 48 percent, which is a significant reduction. Other railroads have used estimated savings such as these to justify projects in the millions of dollars. The adaptive learning model reduces empties to about 35 percent after 25 iterations, with most reductions after this.

Where are these benefits from approximate dynamic programming coming from? There appear to be two sources. First, when a car becomes empty and available at a shipper, it *must* be moved; it cannot be held at the shipper. Cars that become empty early in the week often have to be moved before customer orders have been booked. A common strategy used by railroads is to move cars to classification yards and regional depots where they can be reassigned to customers after they arrive (by this time, customer orders will often have arrived). A myopic model will never send a car to these locations because there are no customer demands at these locations. Using approximate value functions, the model will send cars to locations where they have the highest value, which could easily be a regional depot or classification yard.

The second, and perhaps more significant reason that approximate dynamic programming works is that it captures the uncertain presence of cars in the future. In a snapshot of cars and orders, there is often far more information about orders in the future than cars. A customer might book his order a week or more into the future, but we are generally unable to track cars into the future. We cannot forecast the availability of cars after they fulfill orders that have not even been called in yet. Furthermore, even orders that have been called in are incomplete; shippers do not identify the destination for a car until after the car has been loaded.

The behavior that we have observed is that a myopic model has a tendency to assign a car that is available now to an order that is known now, but does not have to be served until the future. The myopic model is not able to assign a car after it has been used to fulfill an order that is either unknown or has incomplete information (e.g. the destination). The approximate dynamic programming model takes this information implicitly into account when it solves subproblems in the future. Thus, the adaptive learning technology is not only learning about future orders that are currently unknown, but also future cars.

11.1.4 Notes

The car distribution problem captures some of the richness of dynamic information processes that arise in real problems. At the same time, it offers some convenient simplifications. First, we implicitly assume that the attribute spaces \mathcal{A} and \mathcal{B} are fairly small and easy to enumerate. This is not too important in the myopic and rolling horizon models, but is very important if we use the dynamic programming approximation. The reason is that if we act on a car to produce a car with attribute a' , we need to have a value function $\bar{V}_{t,t'a'}^{n-1}$ to measure the value of the resource. Furthermore, we have to update these value functions even if R_{ta} is zero, since an early estimate of the value of a car in the future might have been much lower than what it should be. If $\bar{V}_{t,t'a'}^{n-1}$ underestimates the value of a car of type a' , then we might not implement a decision that produces this car type in the future again. As a result, we have to keep updating the value function to get a good estimate. This has proven to be very important in practice.

In addition, although this is an instance of a two-class resource allocation problem, it is modeled as a one-layer problem. We only model decisions acting on cars, and we only estimate the value of cars in the future (since an order, once it has been moved, is assumed to disappear).

11.2 The dynamic assignment problem for truckload trucking

We next consider the problem of assigning drivers to loads that arises in truckload trucking. We first present the model of the physical process, and then discuss methods for making decisions.

11.2.1 A model of the dynamic load assignment problem

Unlike our car distribution problem, we are going to model this as a two layer problem from the perspective of approximating the value of resources in the future. We also use this problem to illustrate some other modeling devices. For example, in the car distribution problem we represented car types and demand types. This is more natural since there are often a large number of cars (and even orders) of exactly the same type. In the dynamic assignment problem, we are modeling the assignment of “resources” to “tasks” which are discrete in nature (and generally highly individualized). It is more natural, then, to model each resource and task individually. We note that we refer to these as two “resource layers” but use the more common term “resource” to refer to the layer that represents the objects (drivers, trucks, locomotives) and “task” to refer to the customer

or requirement to be satisfied.

We begin by defining:

$$\begin{aligned}\mathcal{C}^R &= \text{Set of resource classes} \\ &= (\text{Drivers(D)}, \text{Loads(L)}).\end{aligned}$$

\mathcal{R}^c = Set of all resource indices for class $c \in \mathcal{C}^R$ that might possibly enter the system.

$$\mathcal{R} = \mathcal{R}^D \cup \mathcal{R}^L.$$

We use d to index elements of the set \mathcal{R}^D and l to index elements of the set \mathcal{R}^L , and we use $r \in \mathcal{R}$ as a generic resource which may be a driver or a load. This convention allows us to avoid defining everything twice.

To describe the state of our system, we define

$$\begin{aligned}R_{t,t'r} &= \begin{cases} 1 & \text{If resource } r \in \mathcal{R} \text{ is known at time } t \text{ and available to be assigned in period } t'. \\ 0 & \text{Otherwise.} \end{cases} \\ R_t &= (R_{t,t'r})_{r \in \mathcal{R}, t' \geq t}.\end{aligned}$$

We use R_t^D and R_t^L to refer specifically to the resource state vector for drivers and loads.

Over time, new drivers and loads will arrive to the system. We assume that there is a master list of identifiers for all drivers and loads (in the set \mathcal{R}), but that there is a specific time at which a driver or load becomes known. We model this process using

$$\begin{aligned}\hat{R}_{t,t'r} &= \begin{cases} 1 & \text{If resource } r \in \mathcal{R}, \text{ which is available to be acted on at time } t', \text{ first becomes} \\ & \text{known in period } t. \\ 0 & \text{Otherwise.} \end{cases} \\ \hat{R}_t &= (\hat{R}_{t,t'r})_{r \in \mathcal{R}, t' \geq t}.\end{aligned}$$

In this simple model, the exogenous information arriving in time period t is given by

$$W_t = (\hat{R}_t^D, \hat{R}_t^L). \tag{93}$$

Our decision problem involves the assignment of drivers to loads, which are represented using

$$\begin{aligned}
x_{tdl} &= \begin{cases} 1 & \text{if driver } d \text{ is assigned to load } l \text{ at time } t. \\ 0 & \text{Otherwise.} \end{cases} \\
x_t &= (x_{tdl})_{d \in \mathcal{R}^D, l \in \mathcal{R}^L} \\
x_{tl}^L &= \sum_{d \in \mathcal{R}^D} x_{tdl} \\
&= \begin{cases} 1 & \text{If any driver is assigned to load } l \text{ at time } t. \\ 0 & \text{Otherwise.} \end{cases} \\
x_{td}^D &= \sum_{l \in \mathcal{R}^L} x_{tdl} \\
&= \begin{cases} 1 & \text{If driver } d \text{ is assigned to any load at time } t. \\ 0 & \text{Otherwise.} \end{cases}
\end{aligned}$$

We use these variables to write our resource dynamics as

$$R_{t+1}^D = R_t^D - x_t^D + \hat{R}_{t+1}^D \quad (94)$$

$$R_{t+1}^L = R_t^L - x_t^L + \hat{R}_{t+1}^L. \quad (95)$$

11.2.2 Optimization formulation

Our challenge, of course, is one of designing a decision function $X_t^\pi(R_t)$. Our goal is to maximize the total contribution from assigning drivers to loads over a planning horizon T^{ph} . Let:

c_{tdl} = The contribution from assigning driver d to load l at time t .

Our one period contribution function is given by

$$C_t(x_t) = \sum_{r \in \mathcal{R}_t^D} \sum_{l \in \mathcal{R}_t^L} c_{tdl} x_{tdl}. \quad (96)$$

We assume that we only receive a contribution when a decision is implemented, which can only occur when both the resource and task are actionable. However, we can *plan* decisions in the future, which is what we assume we are doing when we assign a driver to a load when one or both is actionable in the future. As discussed in section 7 we make decisions by solving a problem of the form

$$X_t^\pi(R_t) = \arg \max_x C_t^\pi(x). \quad (97)$$

Keep in mind that the function X_t^π returns a vector $x_t = (x_{tt'})_{t' \geq t}$, which means it might include actions to be implemented now, x_{tt} , and plans of what to do in the future, $(x_{tt'})_{t' \in \mathcal{T}^{ph} \setminus 0}$. We may limit our attention to resources that are knowable now and actionable now, R_{tt} , or we may include resources that are actionable in the future, $(R_{tt'})_{t' \in \mathcal{T}^{ph}}$. Both represent forms of deterministic models.

An interesting (and perhaps unexpected) behavior arises when we consider resources that are known now but actionable in the future. It would seem natural that such a model would outperform a model that focuses only on resources that are actionable now, but looking further into the future can produce poor results if it is not done well. In most applications, the extent to which we know about internal resources (drivers, trucks, locomotives, freight cars) in the future can be quite different from how far we know about customer orders. In rail and trucking, it is typical for the customers to call in orders farther into the future than we would know about drivers and equipment. For example, it can be hard to track rail equipment into the future because even when a customer order is known, we might not know everything about the order (most obviously, the destination of the order, but also the time required to complete the order). This can make it difficult to track a car into the future for more than a few days. Yet customers will call in orders a week or even two weeks into the future.

Incorporating point forecasts of the future can be difficult for this problem class, since all the activities are (0/1). First, while we have made the explicit choice to model specific drivers and loads, as opposed to driver and load types (as we did in our car distribution model), when we perform forecasting we have to forecast by type (but we can treat a forecasted driver or load as if it were real). Generally, we are not able to forecast as many attributes of a real resource, which means that forecasted resources have to be somewhat simplified. Second, we encounter problems if we have to forecast a type of load that occurs with probability 0.20.

An alternative strategy for handling uncertain, forecasted activities is to resort to value function approximations. Here again, it is more useful to think of estimating the value of a type of driver or load, rather than a specific driver or load. To handle this transition, we let:

a_{tr} = The attributes of resource (driver or load) r at time t .

If we decide to hold a driver (not assign the driver to any load) then this means we have this same driver in the future. This is a “do nothing” decision (which we have represented using d^ϕ) which

might produce a modified set of attributes given by $a^M(t, a_r, d^\phi)$. We note that

$$\begin{aligned} R_{td}^{x,D} &= (1 - x_{td}^D) \\ &= \begin{cases} 1 & \text{If driver } d \text{ is held at time } t \\ 0 & \text{Otherwise.} \end{cases} \end{aligned}$$

Similarly, let $R_{tl}^{x,L}$ indicate whether load l is held until the next time period. $R_t^x = (R_t^{x,D}, R_t^{x,L})$ is our post-decision resource state variable. The decision function is given by

$$X_t^\pi(R_t) = \arg \max_{x \in \mathcal{X}_t} \sum_{d \in \mathcal{R}_t^D} \sum_{l \in \mathcal{R}_t^L} (c_{tdl} x_{tdl} + \bar{V}_t(R_t^x)). \quad (98)$$

subject to appropriate flow conservation constraints on drivers and loads. We then have to determine an appropriate functional form for $\bar{V}_t(R_t)$. The most obvious to choose is a linear approximation

$$\bar{V}_t(R_t) = \sum_{d \in \mathcal{R}_t^D} \bar{v}_{td}^D R_{td}^D + \sum_{l \in \mathcal{R}_t^L} \bar{v}_{tl}^L R_{tl}^L. \quad (99)$$

\bar{v}_{td}^D is the value of driver d if it is held at time t , and \bar{v}_{tl}^L is the value of load l . We note that this approximation is, for the first time, capturing the value of holding both resource layers in the future. We can easily estimate these values directly from the dual variables for the driver and load resource constraints (94) and (95). One complication is that the dual variables, since they are simply subgradients, are typically not accurate enough. If $R_{td}^{x,D} = 1$, then we want the impact of eliminating driver d from the future; if $R_{td}^{x,D} = 0$, then we want the value of adding driver d to the future. It is possible to find these left and right gradients (respectively) by solving flow augmenting path problems into and out of the supersink to all the nodes in the network.

If we introduce our linear approximation into (98), we obtain, after some manipulation

$$X_t^\pi(R_t) = \arg \max_{x \in \mathcal{X}_t} \sum_{d \in \mathcal{R}_t^D} \sum_{l \in \mathcal{R}_t^L} \left(c_{tdl} - \bar{v}_{t,a^M(t,a_d,d^\phi)}^D - \bar{v}_{t,a^M(t,a_l,d^\phi)}^L \right) x_{tdl} \quad (100)$$

where $a^M(t, a_d, d^\phi)$ is the attribute vector of a driver that has been held, and $a^M(t, a_l, d^\phi)$ is the attributes of a load. We note that (100) is still an assignment problem which is an important practical result.

There is still a problem. If $x_{tdl} = 1$, then we are assigning driver d to load l , and *both* are removed from the problem in the future. So, each decision at time t impacts a driver *and* a load. If x_{tdl} is increased from 0 to 1, then we have the effect of decreasing R_{td}^D and R_{tl}^L each by one. This is approximated by $-(\bar{v}_{td}^D + \bar{v}_{tl}^L)$. Needless to say, this will introduce errors. A better approximation would be to capture the nonseparability of the value function. We could, instead, let

$$\bar{v}_{tdl} = \text{The marginal contribution of holding driver } d \text{ and load } l \text{ at the same time.}$$

Of course, we use $-\bar{v}_{tdl}$ to approximate the impact of assigning driver d to load l and therefore eliminating both of them from the pool of resources in the future. We now find ourselves solving subproblems of the form

$$X_t^\pi(R_t) = \arg \max_{x \in \mathcal{X}_t} \sum_{d \in \mathcal{R}_t^D} \sum_{l \in \mathcal{R}_t^L} (c_{tdl} - \bar{v}_{tdl}) x_{tdl}. \quad (101)$$

As before, (101) is still also an assignment problem, which means that if we can estimate \bar{v}_{tdl} , then our single-period problem in equation (98) is no more difficult than the original myopic problem.

Not surprisingly, \bar{v}_{tdl} (see Spivey & Powell (2004) for details) is considerably harder to compute than \bar{v}_{td}^D and \bar{v}_{tl}^L , which require only two flow-augmenting path calculations. \bar{v}_{tdl} , by contrast, requires a flow augmenting from each load node l back to each driver node d . This can be accomplished with a flow-augmenting path for each driver, which is substantially more difficult than the single flow augmenting path into and out of the supersink we required for the separable approximation.

11.2.3 Some numerical experiments

The optimization model allows us to provide answers (experimentally) to two types of questions. First, does an ADP policy (that is, a policy that uses approximate dynamic programming) provide solutions that are better than myopic policies, and how does this compare with the availability of advance information? Second, what is the value of advance information, and how does the value of advance information change with the sophistication of the policy used? These two questions illustrate the two perspectives of dynamic models that we addressed in the introduction.

These questions were addressed in Spivey & Powell (2004). The issue of solution quality was addressed by running a series of simulations. For each simulation, we can compute the optimal

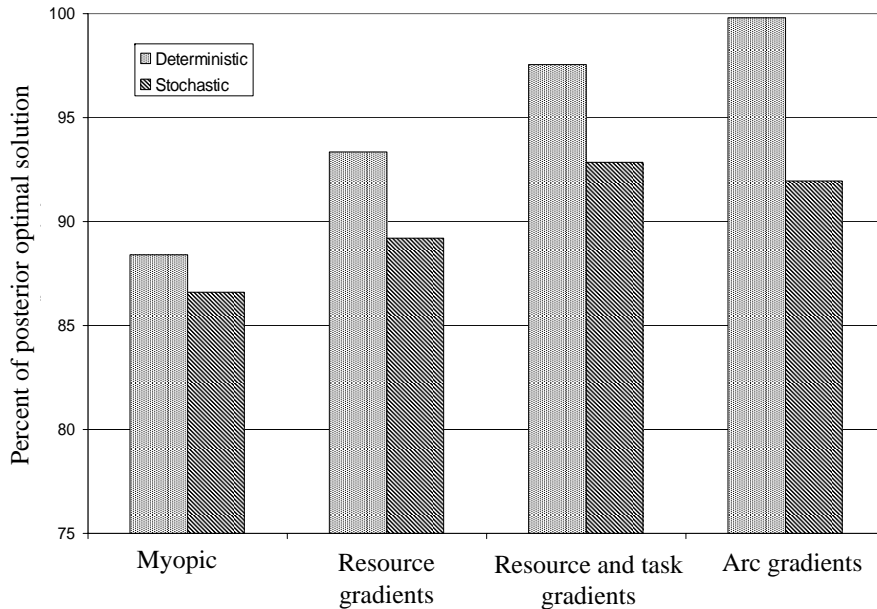


Figure 11: The effect of resource gradients alone, resource and task gradients, and arc gradients, on solution quality for deterministic and stochastic problems

solution after all the information is known (this is simply an assignment problem using all the resources and tasks), giving us a posterior bound. The first set of simulations used deterministic data, which is to say that $|\Omega| = 1$. For this case, we can hope to achieve a solution that is close to the posterior bound. The second set of simulations used a stochastic data set, with $|\Omega| \gg 1$, where there are a number of different outcomes of new drivers and loads.

The results of the simulations are summarized in figure 11, which represent an average over 10 different datasets. For the deterministic datasets, we see a steady improvement as we move from a myopic policy ($\bar{v}^R = \bar{v}^L = 0$), to resource gradients (where we use only \bar{v}^R), to resource and task gradients (where we use \bar{v}^R and \bar{v}^L), to the arc gradients (where we use \bar{v}_{tdl}). It is very encouraging that the arc gradients give near optimal solutions. For stochastic datasets, the best results are obtained when we use resource and task gradients; interestingly, the far more computationally intensive arc gradients do not seem to add any value. It appears that the added precision that these gradients bring are lost in the presence of uncertainty.

It is usually the case in transportation that we will know about resources and tasks in the future, represented by the resource vector $R_{tt'}$. Simulations were run which allowed the decision function to consider resources and tasks that are known now (at time t) but actionable in the future (at

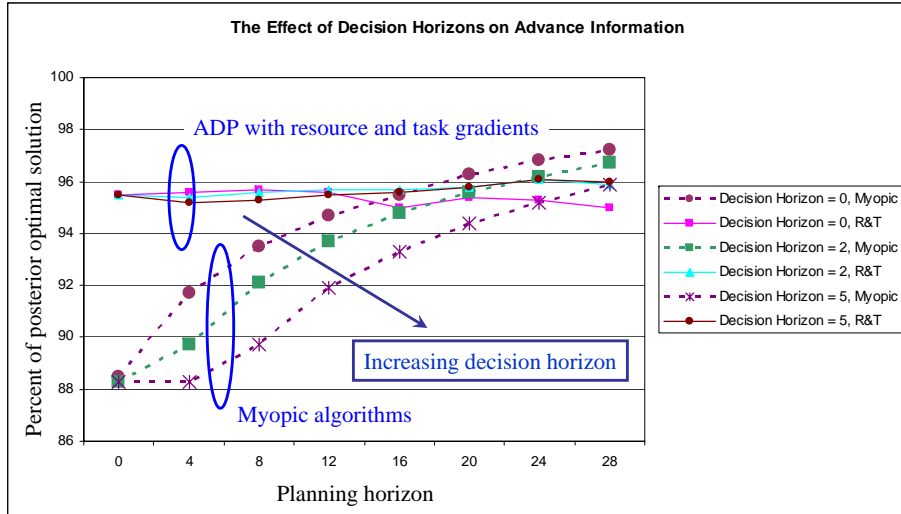


Figure 12: The value of advance information as estimated by a myopic policy, and using approximate dynamic programming (from Spivey & Powell (2004))

time t') as long as the actionable time is within a planning horizon T^{ph} . We would expect to get better results as the planning horizon is increased (which is an indication of the value of advance information) but it is not clear how much this answer is influenced by the use of value function approximations.

Figure 12 shows the improvement in total profits (always measured against the posterior bound) for both myopic and ADP policies. In each case, the policies were tested not only for different values of the planning horizon T^{ph} (the time range over which information is used) but also for different decision horizons (the time period over which decisions in the future are locked into place). The results show that increasing the planning horizon is significant in the case of a myopic policy, but is surprisingly almost nonexistent when we use an ADP policy. At a minimum, it appears that ADP policies benefit less from advance information than a myopic policy. The implication is that measuring the value of advance information using myopic policies might produce inflated estimates.

11.3 Batch processes for less-than-truckload trucking

Up to now, we have considered problems that can be viewed as “flow problems.” Although both the car distribution problem and load matching problem involve discrete resources, these are naturally formed as linear programs with integrality requirements, and they can either be solved as pure networks or linear programs which provide near-integer solutions.

A problem class where this is not the case arises in batch processes such as less-than-truckload trucking, where we have to consolidate smaller packages onto larger vehicles. The decision is how to route the shipments, and when and where to send the trucks. The decision to send a truck has to balance the value of moving the shipments now against various costs of holding the shipments. Since sending a truck has the same mathematical structure as building a link in the network, this has been referred to as the dynamic network design problem (Crainic (2000)).

As of this writing, we are not aware of any computationally tractable algorithms that have proven successful for realistic instances of the dynamic network design problem, at least as it arises in less-than-truckload trucking. Heuristics have been applied with varying degrees of success for static instances (Powell (1986), Crainic & Roy (1988), Crainic & Rousseau (1988)), but the dynamic versions are much harder. Most of the research has focused on solving the problem with a single link. If we face the problem of dispatching a truck over a single link with homogeneous customers, then we have a textbook dynamic program that is easy to solve.

The challenge arises when there is more than one product type. Speranza & Ukovich (1994) and Speranza & Ukovich (1996) develop optimal strategies for the deterministic multiproduct problem. Bertazzi & Speranza (1999*b*) and Bertazzi & Speranza (1999*a*) consider the deterministic problem of shipping products from an origin to a destination through one or several intermediate nodes and compare several classes of heuristics including decomposition of the sequence of links, an EOQ-type solution and a dynamic programming-based heuristic. The results assume that demands are both deterministic and constant. Bertazzi et al. (2000) consider a stochastic, multi-product problem which is solved using approximate dynamic programming techniques, but the method is only tested with a single product type.

In this section, we briefly review the single-link model described in Papadaki & Powell (2003), which considers multiple product types (the method is tested on a problem with 100 product types), uncertainty and significant nonstationarities. For transportation, a “product type” might refer to the destination of the shipment, the time at which the shipment has to arrive at the destination, and the shipment priority (other attributes include size and weight).

11.3.1 A single link model of the dynamic dispatch problem

We begin by defining the following:

Problem parameters

\mathcal{K} = Set of customer classes.

c^d = Cost to dispatch a vehicle.

c_k^h = Holding cost of class $k \in \mathcal{K}$ per time period per unit product.

$c^h = (c_1^h, c_2^h, \dots, c_{|\mathcal{K}|}^h)$

K = Service capacity of the vehicle, giving the total number of customers who can be served in a single dispatch.

Activity variables

R_{tk}^x = Number of customers in class k waiting at time t before new arrivals have been added (the post-decision state variable).

$R_t^x = (R_{tk}^x)_{k \in \mathcal{K}}$

\hat{R}_t = Vector random variable giving the number of arrivals in time t of each type of customer.

$R_t = R_{t-1}^x + \hat{R}_t$

= Number of customers waiting just before we make a decision at time t .

Decision variables

x_{tk} = The number of customers in class k who are served at time t .

$X_t^\pi(R_t)$ = Decision function returning x_t .

Our feasible region \mathcal{X}_t is defined by

$$\begin{aligned} x_{tk} &\leq R_{tk} \\ \sum_{k \in \mathcal{K}} &\leq K. \end{aligned}$$

It is useful to define a dispatch function:

$$Z_t(x_t) = \begin{cases} 1 & \text{If } \sum_{k \in \mathcal{K}} x_{tk} > 0 \\ 0 & \text{Otherwise} \end{cases}$$

Written this way, $x_t = X_t^\pi$ determines Z_t . Later we show that we can actually determine Z_t first, and then compute X_t^π .

Objective function

The one-period cost function is given by

$$C_t(R_t, \hat{R}_t, x_t) = c^d Z_t(x_t) + c^h (R_t - x_t).$$

The objective function over the planning horizon can now be written

$$F(R_0) = \min_{\pi \in \Pi} E \left\{ \sum_{t=0}^T C_t(R_t, X_t^\pi(R_t)) \right\}.$$

11.3.2 A solution algorithm for the single-link problem

The most common practical approach for this problem is to use a simple dispatch rule that might look like “if we have at least a certain amount of freight, send the vehicle, otherwise hold it.” Such rules generally have to be time dependent (“if it is Friday night...”) or state dependent (“if the truck has been held more than 12 hours and it is at least 80 percent full ...”) since the arrival process of freight can be highly nonstationary. Any effort to use optimization has to recognize that such rules can be fairly effective, and at a minimum are easy to understand.

The value of solving this problem with optimization is to impose the discipline of formulating and minimizing a cost function. The problem is that algorithms for network design have generally not been very effective. We propose to solve this problem (which is high dimensional because of the multiple product types) using approximate dynamic programming.

Using our standard methodology, we can formulate a decision function using

$$X_t^\pi(R_t) = \arg \min_{x_t} (C_t(R_t, x_t) + \bar{V}_{t+1}^n(R_t(\omega_t), x_t)). \quad (102)$$

It is possible to show that the value function increases monotonically in the resources; in effect, the more customers that are waiting, the higher the cost. The function itself is neither concave

or convex, but especially when we introduce uncertainty, it is approximately linear, suggesting the approximation

$$\bar{V}_t(R_t) = \bar{v}_t R_t.$$

Before, we estimated our slopes using dual variables from solving the subproblems. In the case of batch processes, we do not have access to dual variables, but we can use finite differences

$$\tilde{v}_{kt} = \tilde{V}_t(R_t + e_k, \omega) - \tilde{V}_t(R_t, \omega)$$

where e_k is a $|\mathcal{K}|$ -dimensional vector with a single 1 in the k^{th} element. Performing these for each product class can be extremely expensive. A short-cut, suggested in Graves (1981) in the context of multistage lot-sizing, is to assume that increasing the number of customers (of any type) does not change the dispatch decision at time t . This makes it trivial to determine the marginal impact of extra customers.

Rather than solve (102) to find x_t , it actually makes more sense to find z_t first (that is, determine whether the batch will be sent or held), and then compute x_t using a simple rule. If $z_t = 0$, then clearly $x_t = 0$. If $z_t = 1$ and $\sum_{k \in \mathcal{K}} R_{tk} \leq K$, then $x_{tk} = R_{tk}$. The only interesting case arises when $z_t = 1$ and $\sum_{k \in \mathcal{K}} R_{tk} > K$. Assume that the customer classes are ordered so that $c_1^h \leq c_2^h \leq \dots \leq c_k^h$. Then, we want to make sure customers in class 1 are all added to the batch before trying to add customers of class 2, and so on, until we reach the capacity of the batch. Assuming we use this rule, the steps of an approximate dynamic programming algorithm are summarized in figure 13.

The algorithm was tested on a single-product problem in Papadaki & Powell (2003) so that comparisons against an optimal solution could be made (using classical dynamic programming). The approximate dynamic programming approach was compared to a rule which specified that the vehicle should be dispatched when full, or if it had been held τ units of time. For each dataset, τ was chosen so that it produced the best results (the assumption being that for any specific problem class, we could choose the best holding time).

The method was tested on a variety of datasets which included both stationary and nonstationary arrival processes. An important determinant of performance is the relative size of the holding cost per unit c^h , and the dispatch cost per unit of capacity c^d/K . For this reason, the datasets were divided into three groups: $c^h > c^d/K$, $c^h \simeq c^d/K$ and $c^h < c^d/K$. In the first group, the

Step 1 Given R_0 : Set $\bar{v}_t^0 = 0$ for all t . Set $n = 1, t = 0$.

Step 2 Set $R_0^n = R_0$ and choose random sample ω^n .

Step 3 Calculate

$$z_t^n = \arg \min_{z_t \in \{0,1\}} \left\{ c^d z_t + c^h \cdot (R_t^n - z_t X(R_t^n)) + \bar{v}_t^n (R_t^n - z_t X(R_t^n)) \right\}$$

and

$$R_{t+1}^n = R_t^n - z_t X(R_t^n) + \hat{R}_{t+1}(\omega^n)$$

Then define:

$$\tilde{V}_t^n (R_t^n) = \min_{z_t \in \{0,1\}} \left\{ c^d z_t + c^h \cdot (R_t^n - z_t X(R_t^n)) + \bar{v}_t^n (R_t^n - z_t X(R_t^n)) \right\}$$

Step 4 Update the approximation as follows. For each $k = 1, \dots, |\mathcal{K}|$, let:

$$\hat{v}_{tk}^n = \tilde{V}_t^n (R_t^n + e_k) - \tilde{V}_t^n (R_t^n)$$

where e_k is an $|\mathcal{K}|$ -dimensional vector with 1 in the k^{th} entry and the rest zero. Update the approximation by smoothing:

$$\bar{v}_t^n = (1 - \alpha^n) \bar{v}_t^{n-1} + \alpha^n \hat{v}_t^n$$

Figure 13: Approximate dynamic programming algorithm for the batch dispatch problem

holding cost is high enough that we will tend to dispatch vehicles very quickly. In the last group, the holding cost is low enough that we expect a dispatch-when-full policy to be best. The middle group is the most interesting.

For each group, we further divided the runs between datasets where the arrival process followed a periodic (e.g. daily) pattern (which was highly nonstationary) from datasets where the arrival process was stationary. The results are reported in table 3, which shows that the approximate dynamic programming approach works better than the optimized myopic heuristic, even for the relatively easier case $c^h < c^d/K$ where a dispatch-when-full strategy tends to work well (this strategy still struggles when the arrival process of customers is highly nonstationary).

Table 4 summarizes the results of runs done with 100 product types. In this case, we cannot solve the problem optimally, but we are able to compare against our myopic policy, and we have an idea

	Method	linear	linear	linear	linear	DWF-
	Iterations	(25)	(50)	(100)	(200)	TC
$h > c/K$	periodic	0.082	0.070	0.058	0.062	0.856
	statioary	0.071	0.050	0.045	0.038	0.691
$h \simeq c/K$	periodic	0.040	0.031	0.024	0.024	0.270
	statioary	0.057	0.035	0.023	0.024	0.195
$h < c/K$	periodic	0.029	0.025	0.019	0.019	0.067
	statioary	0.031	0.019	0.015	0.013	0.059
	Average	0.052	0.038	0.031	0.030	0.356

Table 3: Fraction of total costs produced by each algorithm over the optimal cost: averages and standard deviations within each group (from Papadaki & Powell (2003)).

	Method	adp	adp	adp	adp	adp	adp	adp	adp
		scalar	scalar	scalar	scalar	mult.	mult.	mult.	mult.
	Iterations	(25)	(50)	(100)	(200)	(25)	(50)	(100)	(200)
$h > c/K$	periodic	0.602	0.597	0.591	0.592	0.633	0.626	0.619	0.619
	statioary	0.655	0.642	0.639	0.635	0.668	0.660	0.654	0.650
$h \simeq c/K$	periodic	0.822	0.815	0.809	0.809	0.850	0.839	0.835	0.835
	statioary	0.891	0.873	0.863	0.863	0.909	0.893	0.883	0.881
$h < c/K$	periodic	0.966	0.962	0.957	0.956	0.977	0.968	0.965	0.964
	statioary	0.976	0.964	0.960	0.959	0.985	0.976	0.971	0.969
	Average	0.819	0.809	0.803	0.802	0.837	0.827	0.821	0.820

Table 4: The (expected) cost of the approximate dynamic programming algorithm as a fraction of the cost of the DWF-TC myopic heuristic for both scalar (single product) and multiple product problems (from Papadaki & Powell (2003)).

from the single product case of how well the myopic policy works. In this table, instead of reporting the percent over optimal, we report the total costs of the approximate dynamic programming policy divided by the total costs of the myopic policy. If we again focus on the class of problems where $h < c/K$ (where the myopic policy will work the best) we find that approximate dynamic programming strategies are approximately three percent better than the myopic policy, which is very consistent with our results from the single product case.

12 Perspectives on real-time problems

Dynamic models have a number of applications. We can simulate a dynamic process to better understand how to operate a system under more realistic settings. We might want to investigate the value of better information (through automated detection systems, better databases, processes that require customers to make requests known further into the future). A company might want to produce short term tactical forecasts (for example, to help identify bottlenecks) one or two days into the future given the state of the system right now.

One application that often arises at many companies is a desire to develop models to help run their operations in real time. While there are a number of ways to help a company make better decisions now (for example, with short term tactical forecasts), there is often an interest in having computers tell dispatchers and planners what to do (that is, to automate the decision itself).

There is an important, qualitative difference between a real-time, dynamic model that is used to look into the future to produce forecasts, and a real-time, dynamic model that is used to make specific, actionable recommendations. The first is trying to forecast activities in the future to help a human make better decisions now. The second, which is much harder, is actually trying to tell a human what to do right now. True real-time optimization can be viewed as the information-age version of factory robots. We anticipate that these will steadily make their way into operations, but adoption will be slow. Sometimes it is only when we try to replace a human that we fully appreciate the diversity of tasks that people perform.

References

- Aronson, J. & Chen, B. (1986), ‘A forward network simplex algorithm for solving multiperiod network flow problems’, *Naval Research Logistics Quarterly* **33**(3), 445–467.
- Aronson, J. & Thompson, G. L. (1984), ‘A survey on forward methods in mathematical programming’, *Large Scale Systems* **7**, 1–16.
- Assad, A. (1978), ‘Multicommodity network flows: A survey’, *Networks* **8**(1), 37–91.
- Baker, S., Morton, D., Rosenthal, R. & Williams, L. (2002), ‘Optimizing military airlift’, *Operations Research* **50**(4), 582–602.
- Barnhart, C., Hane, C. A., Johnson, E. L. & Sigismondi, G. (1998), ‘Branch-and-price: Column generation for solving huge integer programs’, *Operations Research* **46**(3), 316–329.
- Bellman, R. (1957), *Dynamic Programming*, Princeton University Press, Princeton.
- Bellman, R. & Dreyfus, S. (1959), ‘Functional approximations and dynamic programming’, *Mathematical Tables and Other Aids to Computation* **13**, 247–251.

- Bertazzi, L. & Speranza, M. G. (1999a), ‘Inventory control on sequences of links with given transportation frequencies’, *International Journal of Production Economics* **59**, 261–270.
- Bertazzi, L. & Speranza, M. G. (1999b), ‘Minimizing logistic costs in multistage supply chains’, *Naval Research Logistics* **46**, 399–417.
- Bertazzi, L., Bertsekas, D. & Speranza, M. G. (2000), Optimal and neuro-dynamic programming solutions for a stochastic inventory transportation problem, Unpublished technical report, Università Degli Studi Di Brescia.
- 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.
- 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.
- Cinlar, E. (2003), Private communication.
- Crainic, T. (2000), ‘Network design in freight transportation’, *European Journal of Operational Research* **12**(2), 272–288.
- Crainic, T. & Rousseau, J.-M. (1988), ‘Multicommodity, multimode freight transportation: A general modeling and algorithmic framework for the service network design problem’, *Transportation Research B* **20B**, 290–297.
- Crainic, T. & Roy, J. (1988), ‘OR tools for the tactical planning of freight transportation’, *European Journal of Operations Research* **33**, 290–297.
- Desrochers, M. & Soumis, F. (1989), ‘A column generation approach to the urban transit crew scheduling problem’, *Transportation Science* **23**, 1–13.
- Desrosiers, J., Solomon, M. & Soumis, F. (1995), Time constrained routing and scheduling, in C. Monma, T. Magnanti & M. Ball, eds, *Handbook in Operations Research and Management Science, Volume on Networks*, North Holland, Amsterdam, pp. 35–139.
- Desrosiers, J., Soumis, F. & Desrochers, M. (1984), ‘Routing with time windows by column generation’, *Networks* **14**, 545–565.
- Ford, L. & Fulkerson, D. (1962), *Flows in Networks*, Princeton University Press, Princeton.
- Glockner, G. D. & Nemhauser, G. L. (2000), ‘A dynamic network flow problem with uncertain arc capacities: Formulation and problem structure’, *Operations Research* **48**, 233–242.
- Glover, F., Karney, D., Klingman, D. & Napier, A. (1974), ‘A computation study on start procedures, basis change criteria, and solution algorithms for transportation problems’, *Management Science* **20**, 793 – 813.
- Glover, F., Klingman, D. & Phillips, N. V. (1992), *Network Models in Optimization and their Application in Practice*, John Wiley & Sons, Inc., New York.
- Godfrey, G. & Powell, W. B. (2002), ‘An adaptive, dynamic programming algorithm for stochastic resource allocation problems I: Single period travel times’, *Transportation Science* **36**(1), 21–39.

- 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.
- Graves, S. C. (1981), Multi-stage lot sizing: An iterative procedure, in L. B. Schwarz, ed., ‘Multi-level Production/Inventory Control Systems: Theory and Practice’, Vol. 16 of *TIMS Studies in the Management Sciences*, North-Holland Publishing Company, New York, pp. 95–110.
- Infanger, G. (1994), *Planning under Uncertainty: Solving Large-scale Stochastic Linear Programs*, The Scientific Press Series, Boyd & Fraser, New York.
- Kall, P. & Wallace, S. (1994), *Stochastic Programming*, John Wiley and Sons, New York.
- Kennington, J. & Helgason, R. (1980), *Algorithms for Network Programming*, John Wiley & Sons, Inc., New York.
- Kennington, J. L. (1978), ‘A survey of linear cost multicommodity network flows’, *Operations Research* **26**, 209–236.
- Kushner, H. J. & Yin, G. G. (1997), *Stochastic Approximation Algorithms and Applications*, Springer-Verlag, New York.
- Langley, R. W., Kennington, J. L. & Shetty, C. M. (1974), ‘Efficient computational devices for the capacitated transportation problem’, *Naval Research Logistics Quarterly* **21**, 637–647.
- Lasdon, L. (1970), *Optimization Theory for Large Systems*, MacMillan, New York.
- Lavoie, S., Minoux, M. & Odier, E. (1988), ‘A new approach of crew pairing problems by column generation and application to air transport’, *European Journal of Operational Research* **35**, 45–58.
- Marar, A. & Powell, W. B. (2004), Using static flow patterns in time-staged resource allocation problems, Technical report, Princeton University, Department of Operations Research and Financial Engineering.
- Marar, A., Powell, W. B. & Kulkarni, S. (to appear), ‘Capturing expert knowledge in resource allocation problems through low-dimensional patterns’, *IIE Transactions*.
- Morton, D. P., Rosenthal, R. E. & Lim, T. W. (1996), ‘Optimization modeling for airlift mobility’, *Military Operations Research* pp. 49–67.
- Morton, D. P., Salmeron, J. & Wood, R. K. (2002), ‘A stochastic program for optimizing military sealift subject to attack’, *Stochastic Programming e-print series*. <http://www.speps.info>.
- Papadaki, K. & Powell, W. B. (2003), ‘An adaptive dynamic programming algorithm for a stochastic multiproduct batch dispatch problem’, *Naval Research Logistics* **50**(7), 742–769.
- Powell, W. B. (1986), ‘A local improvement heuristic for the design of less-than-truckload motor carrier networks’, *Transportation Science* **20**(4), 246–257.
- 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., Shapiro, J. A. & Simão, H. P. (2001), A representational paradigm for dynamic resource transformation problems, in R. F. C. Coullard & J. H. Owens, eds, ‘Annals of Operations Research’, J.C. Baltzer AG, pp. 231–279.
- Powell, W. B., Shapiro, J. A. & Simão, H. P. (2002), ‘An adaptive dynamic programming algorithm for the heterogeneous resource allocation problem’, *Transportation Science* **36**(2), 231–249.
- Powell, W. B., Wu, T. T. & Whisman, A. (2004), ‘Using low dimensional patterns in optimizing simulators: An illustration for the airlift mobility problem’, *Mathematical and Computer Modeling* **29**, 657–2004.

- 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.
- Sen, S. & Higle, J. (1999), ‘An introductory tutorial on stochastic linear programming models’, *Interfaces* **29**(2), 33–61.
- Shapiro, J. & Powell, W. B. (to appear), ‘A metastrategy for dynamic resource management problems based on informational decomposition’, *Inform Journal on Computing*.
- Speranza, M. & Ukovich, W. (1994), ‘Minimizing transportation and inventory costs for several products on a single link’, *Operations Research* **42**, 879–894.
- Speranza, M. & Ukovich, W. (1996), ‘An algorithm for optimal shipments with given frequencies’, *Naval Research Logistics* **43**, 655–671.
- Spivey, M. & Powell, W. B. (2004), ‘The dynamic assignment problem’, *Transportation Science* **38**(4), 399–419.
- Sutton, R. (1988), ‘Learning to predict by the methods of temporal differences’, *Machine Learning* **3**, 9–44.
- Sutton, R. & Barto, A. (1998), *Reinforcement Learning*, The MIT Press, Cambridge, Massachusetts.
- Topaloglu, H. & Powell, W. B. (to appear), ‘Dynamic programming approximations for stochastic, time-staged integer multicommodity flow problems’, *Inform Journal on Computing*.
- Tsitsiklis, J. & Van Roy, B. (1997), ‘An analysis of temporal-difference learning with function approximation’, *IEEE Transactions on Automatic Control* **42**, 674–690.
- Van Roy, B. (2001), Neuro-dynamic programming: Overview and recent trends, in E. Feinberg & A. Shwartz, eds, ‘Handbook of Markov Decision Processes: Methods and Applications’, Kluwer, Boston.
- Vance, P. H., Barnhart, C., Johnson, E. L. & Nemhauser, G. L. (1997), ‘Airline crew scheduling: A new formulation and decomposition algorithm’, *Operations Research* **45**(2), 188–200.
- White, W. (1972), ‘Dynamic transshipment networks: An algorithm and its application to the distribution of empty containers’, *Networks* **2**(3), 211–236.
- Wu, T. T., Powell, W. B. & Whisman, A. (2003), The optimizing simulator: an intelligent analysis tool for the airlift mobility problem, Technical report, Princeton University, Department of Operations Research and Financial Engineering.