Approximate Dynamic Programming for High-Dimensional Resource Allocation Problems

Warren B. Powell
Department of Operations Research and Financial Engineering
Princeton University

Benjamin Van Roy
Departments of Management Science and Engineering and Electrical Engineering
Stanford University

January 23, 2003

Abstract

The allocation of human and physical resources over time is a fundamental problem that is central to management science. In this paper we review a mathematical model of dynamic resource allocation that is motivated by problems in transportation and logistics. In principle, problems of this type can be solved via dynamic programming. However, three "curses of dimensionality" give rise to intractable computational requirements. We present computationally efficient *approximate dynamic programming* algorithms developed by the first author and collaborators for application to problems in freight transportation. We discuss how these algorithms address the three curses of dimensionality and how they relate to other independent threads of research on mathematical programming and approximate dynamic programming.

1 Introduction

The allocation of human and physical resources over time is a fundamental problem that is central to management science. For example, a freight transportation company must manage personnel and equipment to move shipments in a timely manner in the presence of a variety of dynamic information processes: customer demands, equipment failures, weather delays, and failures of execution. This is a high-dimensional problem since it involves a large number of resources, each of which must be tracked as it is affected by decisions and uncertainties.

In principle, problems of dynamic resource allocation can be treated as Markov decision processes and solved using dynamic programming algorithms. Textbook dynamic programming algorithms – such as value iteration and policy iteration – typically require compute time and memory that grow exponentially in the number of state variables, the number of decision variables, and the number of random variables that affect the system in each time period. These three "curses of dimensionality" render such algorithms infeasible for problems of practical scale.

In this paper, we focus on a formulation of dynamic resource allocation that was originally motivated by problems in transportation but also captures problems arising in a variety of other settings. Practical problems formulated in terms of our model typically involve thousands of state variables that describe current resources, thousands of decision variables that determine what is to be done with each resource, and thousands of random variables that influence the state. These random variables capture uncertainties from a variety of sources, such as customer demands, the physical network, and characteristics of the people and equipment used to provide services (e.g., equipment breakdowns and no-shows). Clearly, textbook dynamic programming algorithms are inapplicable, and solving such large-scale dynamic resource allocation problems has proven to be a terrific challenge. In this chapter, we present examples of approximate dynamic programming algorithms developed by the first author and collaborators to address such problems. We also discuss the relation between these algorithms and ideas studied in the broader approximate dynamic literature (including "neuro-dynamic programming" and "reinforcement learning" methods) and the mathematical programming literature.

This chapter is organized as follows. Section 2 presents a mathematical model for a broad class of resource allocation problems together with a simple illustrative example. Section 3 discusses how – in the context of our dynamic resource allocation model – the three curses of dimensionality prevent application of textbook dynamic programming algorithms. In section 4, we present approximate dynamic programming algorithms designed for our model. We discuss in section 5 models and algorithms that have emerged from the field of mathematical programming, and then review in section 6 relations to ideas that have evolved from within the approximate dynamic programming literature. In section 7, we present some experimental results.

2 Dynamic Resource Allocation

In this section, we present a formulation of dynamic resource allocation problems. After defining the formulation in section 2.1, we discuss in section 2.2 a simple example involving the management of a fleet of trucks. This example is a dramatically simplified version of a real trucking problem. However, it serves to illustrate how the formulation maps to a practical context while avoiding the intricacies of truly realistic models (see, for example, Powell (1996) and Powell et al. (2002*b*)).

2.1 Problem Formulation

We consider a system that evolves in discrete time over T periods. At each time $t = 0, \ldots, T$, the state of the system is described by a state variable R_t, taking the form:

$$\begin{aligned} a &= \text{The vector of attributes that describe a single resource.} \\ \mathcal{A} &= \text{The set of possible attributes.} \\ R_{ta} &= \text{The number of resources with attribute } a \text{ at time } t. \\ R_t &= (R_{ta})_{a \in \mathcal{A}}. \end{aligned}$$

For our purposes, it is enough to describe a single resource class such as trucks or people or product, but more complex problems can exhibit multiple resource classes (tractors, trailers and drivers; pilots, aircraft and passengers). For these problems, we have to define different resource classes $\mathcal{C}^R$. We then let $\mathcal{A}^c$ be the attribute space for a resource class $c \in \mathcal{C}^R$, and we let R_t^c be the vector of resources for class c. If we let $R_t = (R_t^c)_{c \in \mathcal{C}^R}$ then we can continue to let R_t be our resource state.

At each time $t = 0, \ldots, T-1$, a decision is made on what to do with each resource. For any resource with attribute a, there is a set $\mathcal{D}_a$ of actions that can be applied to the resource. The collection of actions applied at time t is represented by a decision $x_t = (x_{tad})_{a \in \mathcal{A}, d \in \mathcal{D}_a}$, where x_{tad} is the number of resources with attribute a that action d is applied to. In a transportation example involving the management of equipment, a would capture the type and location of a piece of equipment and an action might be to move the equipment from one location to another. Needless to say, each decision variable x_{tad} must be a nonnegative integer and the decision variables must satisfy flow conservation constraints:

$$\sum_{d \in \mathcal{D}_a} x_{tad} = R_{ta} \quad \forall a \in \mathcal{A} \tag{1}$$

which we write compactly as $A_t x_t = R_t$, where A_t is a linear operator. There may also be additional constraints on the decision variable. We assume that these constraints are linear, taking the form

$$U_t x_t \leq u_t, \tag{2}$$

where U_t is a linear operator mapping decisions to R^{n_t} for some n_t, so that $U_t x_t$ and u_t are elements of $\Re^{n_t}$.

The change that a decision induces on the state R_t is a linear function of the decision. We define a linear operator Δ_t by:

$$(\Delta_t x_t)_a = \text{The number of resources that result in attribute } a \text{ given the decision vector } x_t.$$

The vector R_t of resources that we can act on in time period t is known as the *pre-decision state vector*. The resource vector after the resources have been acted on is given by $R_t^x = \Delta_t x_t$. We refer to R_t^x as the *post-decision* state vector, since it is the state of our system immediately after decisions have been made.

There are also random shocks – new resources appear and existing resources can disappear randomly. This randomness is driven by a sequence $\omega_1, \ldots, \omega_T$ of independent identically distributed random variables. The random change during the tth period in the number of resources with attribute a takes the form $\hat{R}_a(R_t^x, \omega_{t+1})$ for some function $\hat{R}_a$. Note that this quantity is a function of both the random variable ω_{t+1} and the "post-decision" state $R_t^x = \Delta_t x_t$. Let $\hat{R}(R_t^x, \omega_{t+1}) = (\hat{R}_a(R_t^x, \omega_{t+1}))_{a \in \mathcal{A}}$. The dynamics of our system are described by the simple equation:

$$R_{t+1} = \Delta_t x_t + \hat{R}(R_t^x, \omega_{t+1}).$$

If our random changes are purely exogenous arrivals to the system (which do not depend on the state of the system), we will write $\hat{R}(\omega_t)$ as the exogenous arrivals to the system during time interval t.

At each time, we must select a feasible action x_t. For shorthand, we denote the feasible set by $\mathcal{X}_t(R_t)$ – this is the set of actions x_t with nonnegative integer components that satisfy (1) and (2). Each action x_t is selected based only on the current state R_t. Hence, we can think of decisions as being made by a *policy*, which is a function X_t^π that maps each state to a feasible action:

$$x_t = X_t^\pi(R_t) \in \mathcal{X}_t(R_t).$$

There are, of course, many different policies (functions) that we can use, so we let Π be our family of policies (decision functions) that we can choose from.

The contribution (or reward) generated in each tth period is a linear function $C_t x_t$ of the decision x_t. Here, C_t is a linear operator that gives the total contribution (if we are maximizing) of an action. Our objective is to select a policy that maximizes the expected contribution over the horizon:

$$\max_{\pi \in \Pi} E \left[\sum_{t=0}^{T-1} C_t X_t^\pi(R_t) \right].$$

2.2 A Transportation Example

As an example to illustrate how our formulation might map to a real-world context, we discuss in this section a simplified model of a truckload motor carrier. For the sake of brevity, we will only discuss at a very high level how elements of the model might be used to capture features of the application.

Consider the management of N trucks moving among L cities over T time periods. Loads materialize at various cities, and each load is tagged with a destination. For the sake of simplicity, we assume that the only source of uncertainty is in demand (i.e., the loads that materialize in various cities).

We now discuss how one might formulate the problem in terms of our mathematical model. The attribute a specifies either that the resource is residing in a particular city, or that the resource is travelling, in which case the attribute a further specifies the next city where the resource will arrive and the time until arrival.

The random variable $\hat{R}(R_t^x, \omega_{t+1})$ captures new loads entering the system. The number of loads that materialize at each city in a given time period is independent of the numbers at other cities, and these arrivals are independent and identically distributed over time. Note that, in this special case, $\hat{R}$ only depends on ω_{t+1} and not R_t^x.

The set of possible actions $\mathcal{D}_a$ that can be applied to a resource depends on its current attribute. For example, a driver in one location can typically be assigned to move loads that are not too far away. Furthermore, the attribute vector might capture other characteristics of a driver that limit what a driver can do. For example, a Canadian-based driver in the United States can only accept a load that returns him to Canada.

Some additional constraints are placed on the choice of the aggregate collection of actions x_t. The flow conservation constraint (1) ensures that we only act on the trucks and loads that are available. Further requirements are that a load can only move if it is on a truck and that a truck can only carry a single load at a time. These requirements can be represented as a linear constraint $U_t x_t \leq u_t$ for appropriately defined U_t and u_t.

The operator Δ_t captures how resources migrate in the system and how loads leave the system as consequences of decisions. Each load generates a contribution upon delivery. The contribution is a decreasing function of the time taken to serve the load, as specified by its attribute a. The net contribution in a time period can be represented as a linear function $C_t x_t$ of x_t.

3 The Curses of Dimensionality

The problem of dynamic resource allocation can – in principle – be addressed via dynamic programming. In particular, value functions $V_0, \ldots, V_T$ are computed by setting

$$V_T(R_T) = 0, \tag{3}$$

and applying the recursion

$$V_t(R_t) = \max_{x \in \mathcal{X}_t(R_t)} \left(C_t x + E_t \left[V_{t+1} \left(R_t^x + \hat{R}(R_t^x, \omega_{t+1}) \right) \right] \right), \tag{4}$$

where the expectation is taken over possible outcomes of $\hat{R}_{t+1}$. We use the subscript with the expectation operator to indicate the conditioning information. In particular, for any t, E_t denotes a conditional expectation, conditioned on $\omega_1, \ldots, \omega_t$ (if $t = 0$, this translates to an expectation with respect to the prior distribution). Optimal decisions x_t can then be generated according to

$$x_t \in \operatorname*{argmax}_{x \in \mathcal{X}_t(R_t)} \left(C_t x + E_t \left[V_{t+1} \left(R_t^x + \hat{R}(R_t^x, \omega_{t+1}) \right) \right] \right).$$

Three computational obstacles prevent use of this textbook approach when dealing with problems of practical scale. First, the number of possible state vectors R_t grows very quickly with the number $|\mathcal{A}|$ of possible attributes, making computation of $V_t(R_t)$ for every possible R_t unmanageable. Second, exact computation of the expectation is infeasible because the number of possible

outcomes of ω_{t+1} typically becomes enormous as $|\mathcal{A}|$ grows. Finally, finding an optimal decision x_t from the discrete space $\mathcal{X}_t$ generally requires an exhaustive search over $\mathcal{X}_t$. This space once again grows extremely quickly as $|\mathcal{A}|$ grows, rendering an exhaustive search infeasible. We refer to these three obstacles as the three "curses of dimensionality."

We note that some authors have studied formulations similar to what we have presented and solved problem instances using exact dynamic programming techniques (Alden & Smith (1992), Kleywegt & Papastavrou (1998)). However, the problems instances treated involve very few variables – a far stretch from the scale of real transportation systems.

4 Algorithms for Dynamic Resource Allocation

In this section, we present algorithms developed by the first author with collaborators to address large-scale dynamic resource allocation problems of the type presented in section 2. These algorithms incorporate several ideas, one of which involves a reformulation of dynamic programming equations in terms of a "post-decision state." We discuss this reformulation in section 4.1. The algorithms approximate a value function. This requires design of an approximation architecture and execution of algorithms that fit the approximation architecture to the value function. Two particular approximation architectures that have been successfully applied to transportation problems are discussed in section 4.2. In section 4.3 we present algorithms used to fit these approximation architectures.

4.1 The Post-Decision State

Earlier we introduced the notion of a "post-decision state" $R_t^x = R_t + \Delta_t x_t$. This is the state of the system after being modified by the action x_t taken at time t but before being affected by the random variable $\hat{R}_{t+1}$. Let $V_t^x(R_t^x)$ be the maximal expected sum of contributions to be received from decisions $x_{t+1}, \ldots, x_{T-1}$, given the post-decision state R_t^x, so that

$$V_t^x(R_t^x) = E_t\left[V_t\left(R_t^x + \hat{R}(R_t^x, \omega_{t+1})\right)\right].$$

It is easy to show that V_t^x can be computed by setting

$$V_{T-1}^x(R_{T-1}^x) = 0, \tag{5}$$

and applying the recursion

$$V_t^x(R_t^x) = E_t\left[\max_{x \in \mathcal{X}_{t+1}(R_t^x + \hat{R}(R_t^x, \omega_{t+1}))} \left(C_{t+1}x + V_{t+1}^x\left(\Delta_{t+1}x\right)\right)\right]. \tag{6}$$

Further, optimal decisions can be generated according to

$$x_t \in \operatorname*{argmax}_{x \in \mathcal{X}_t(R_t)} \left(C_t x + V_t^x(\Delta_{t+1}x)\right).$$

The new recursion is similar to the standard dynamic programming recursion (4) used for computing V_t. An important difference, however, is that this new recursion (6) involves the expectation of a maximum whereas (4) requires computing the maximum of an expectation. This difference in the form of (6) can be critical for use of simulation-based methods, as we will discuss in the section 4.3.

4.2 Approximation Architectures

Our approach to high-dimensional dynamic resource allocation problems involves approximating each value function V_t^x using an *approximation architecture*; that is, a family of functions $\bar{V}^x$ parameterized by a vector v of real values, so that each $\bar{V}^x(\cdot, v)$ is a function mapping states to real values. Approximation of the value function involves choosing an appropriate approximation architecture and then computing parameters v_t so that $\bar{V}^x(\cdot, v_t) \approx V_t^x$. Given such an approximation, suboptimal decisions x_t can be generated according to

$$x_t \in \underset{x \in \mathcal{X}_t(R_t)}{\operatorname{argmax}} \left(C_t x + \bar{V}^x(\Delta_t x, v_t) \right). \tag{7}$$

When choosing an approximation architecture, one must strike a balance between computational efficiency and performance of the resulting policy. Some approximation architectures may amplify computational challenges. For example, a second or third order polynomial approximation to V_t^x may turn the maximization problem in (7) into a very difficult integer program. On the other hand, specially structured approximation architectures such as those we will discuss next may sometimes lead to computationally tractable integer programming problems; e.g., in some cases the maximization problem in (7) may become an easily solved network optimization problem.

Two classes of approximation architectures have been successfully applied to a variety of problems in transportation operations. These are linear (in the state variable) architectures:

$$\bar{V}^x(R_t, v_t) = \sum_{a \in \mathcal{A}} v_{ta} R_{ta}, \tag{8}$$

and separable concave architectures:

$$\bar{V}^x(R_t, v_t) = \sum_{a \in \mathcal{A}} \left(\sum_{i=1}^{\lfloor R_{ta} \rfloor} v_{tai} + (R_{ta} - \lfloor R_{ta} \rfloor) v_{ta\lceil R_{ta} \rceil} \right), \tag{9}$$

where $v_{tai} \geq v_{taj}$ for all t, a, and $i \leq j$. In the linear case, the parameter vector takes the form $v_t = (v_{ta})_{a \in \mathcal{A}}$, whereas in the separable concave case, we have $v_t = (v_{tai})_{a \in \mathcal{A}, i \in \{1, R_{ta}^{\max}\}}$, where $R_{ta}^{\max}$ is an upper bound on the values that R_{ta}^x can take on. Though states are integer-valued, these approximation architectures assign values to all elements of the positive orthant. This is required to make the optimization problem (7) amenable to available software for solving integer programs. Note that the separable concave architecture can realize any separable concave function on the integers. Each non-integer point is assigned a value corresponding to a linear interpolation. Each of the two architectures offers specific advantages and disadvantages, as we now discuss.

Not surprisingly, linear approximations are the easiest to work with. They will work well when the value function is approximately linear over the range of interest, and also for discrete routing and scheduling problems where $R_{ta} \in \{0, 1\}$. If the one-period problem,

$$x_t \in \underset{x \in \mathcal{X}_t(R_t)}{\operatorname{argmax}} C_t x,$$

exhibits structure that facilitates efficient solution (for example, network or near-network structure) a linear approximation to the value function will retain this structure in (7). Additionally, linear approximations offer computational advantages with regards to algorithms for computing appropriate parameters v_t. Linear approximations should generally be tried before other, more complex strategies, but they can sometimes work poorly. In such cases, one must move on to a richer approximation architecture.

For some problems, the value function is nonlinear but concave, and separable concave functions have proven to offer useful approximations. Although they are typically harder to work with than linear approximations, the decision optimization problem (7) often still retains network or near-network structure if the one-period problem exhibits such structure. For example, in certain problems involving management of homogeneous resources, the optimization problem (7) exhibits pure network structure so that solution of the linear programming relaxation naturally produces integer solutions (see, for example, Godfrey & Powell (2002)). In more realistic problems, resources are heterogeneous. The resulting optimization problems do not generally exhibit pure network structure. Nevertheless, solution of the linear programming relaxation results in integer solutions the vast majority of the time, and non-integer solutions are often easy to resolve.

4.3 Algorithms

In this section, we describe algorithms for computing parameter values to fit linear and separable concave approximations to the value function. These algorithms are representative of ideas developed by the first author and collaborators to tackle large-scale problems in transportation and logistics. In order to deliver a brief and accessible exposition, we have chosen to present relatively simple versions of the algorithms. More complicated variations that improve on execution time and memory requirements can be found in (Powell et al. (2002*b*), Godfrey & Powell (2002), Spivey & Powell (to appear),and Powell & Topaloglu (2003)). Such improvements are often critical for practical application to realistic large-scale problems.

The algorithms we will present are iterative and integrate integer programming, stochastic approximation, and function approximation. Each iteration involves an independent simulated sequence $\omega_1, \ldots, \omega_T$ and adjusts the parameters $v_0, \ldots, v_{T-2}$ by small amounts. The algorithms for linear and separable concave architectures are identical except for one step, so we describe them as one algorithm and only delineate the two cases when required. In stating the algorithm, we will use some new notation. For each $a, \overline{a} \in \mathcal{A}$, let $\delta_{\overline{a}}^{\overline{a}} = 1$ and $\delta_a^{\overline{a}} = 0$ for $a \neq \overline{a}$ (note that this can be thought of as a Dirac delta function). Also, let $\delta^{\overline{a}} = (\delta_a^{\overline{a}})_{a \in \mathcal{A}}$. The algorithm takes as a parameter a step size γ, which is generally chosen to be a small positive scalar. Let each element of v_t be initialized to 0. Each iteration of the algorithm executes the following steps:

1. **Generate random sequence.** Sample the random sequence $\omega_1, \ldots, \omega_T$.

2. **Simulate state trajectory.** For $t = 0, \ldots, T-1$, let

$$\begin{aligned} x_t &\in \underset{x \in \mathcal{X}_t(R_t)}{\operatorname{argmax}} \left(C_t x + \bar{V}^x(\Delta_t x, v_t)\right) \\ R_t^x &= \Delta_t x_t \\ R_{t+1} &= R_t^x + \hat{R}(R_t^x, \omega_{t+1}). \end{aligned}$$

3. **Estimate gradient.** For $t = 0, \ldots, T-2$ and $a \in \mathcal{A}$ such that $R_{ta}^x < R_{ta}^{\max}$, let

$$\begin{aligned} q_t &= C_{t+1} x_{t+1} + \bar{V}^x(R_{t+1}^x, v_{t+1}) \\ R_{ta}^{x+} &= R_t^x + \delta_a \\ q_{ta}^+ &= \max_{x \in \mathcal{X}_{t+1}(R_t^x + \delta^a + \hat{R}(R_t^x + \delta^a, \omega_{t+1}))} \left(C_{t+1} x + \bar{V}^x(\Delta_{t+1} x, v_{t+1})\right) \\ d_{ta} &= q_{ta}^+ - q_t. \end{aligned}$$

4. **Update approximation architecture parameters.** For $t = 0, \ldots, T-2$ and $a \in \mathcal{A}$ such that $R_{ta}^x < R_{ta}^{\max}$, apply

   (a) **Linear case.**

   $$v_{ta} := (1-\gamma) v_{ta} + \gamma d_{ta}.$$

   (b) **Separable concave case.** For the slope element $i = (R_{ta}^x + 1)$:

   $$v_{tai} := (1-\gamma) v_{tai} + \gamma d_{ta},$$

   which may produce a nonconcave function. Restore concavity according to:

$$\begin{aligned} &\text{if } v_{ta(R_{ta}^x+1)} > v_{taR_{ta}^x}: && v_{taR_{ta}^x} := \tfrac{v_{taR_{ta}^x} + v_{ta(R_{ta}^x+1)}}{2} \\ & && v_{ta(R_{ta}^x+1)} := \tfrac{v_{taR_{ta}^x} + v_{ta(R_{ta}^x+1)}}{2} \\ &\text{if } v_{ta(R_{ta}^x+1)} < v_{ta(R_{ta}^x+2)}: && v_{ta(R_{ta}^x+1)} := \tfrac{v_{ta(R_{ta}^x+1)} + v_{ta(R_{ta}^x+2)}}{2} \\ & && v_{ta(R_{ta}^x+1)} := \tfrac{v_{ta(R_{ta}^x+1)} + v_{ta(R_{ta}^x+2)}}{2}. \end{aligned} \tag{10}$$

Let us now provide an interpretation of the algorithm. Each sampled sequence $\omega_1, \ldots, \omega_T$ is used in Step 2 to generate a sample trajectory of states $R_0, R_0^x, \ldots, R_{T-1}, R_{T-1}^x, R_T$. In addition to the random sequence, this state trajectory is influenced by decisions $x_0, \ldots, x_{T-1}$. As the state trajectory is generated, these decisions are selected to optimize an estimate of future contribution. This estimate is taken to be the sum $C_t x_t + \bar{V}^x(R_t + \Delta_t x_t, v_t)$ of immediate contribution and an approximation of subsequent contribution given by $\bar{V}^x(\cdot, v_t)$. Implicit in the statement of our algorithm is that we have access to a subroutine for optimizing this estimate over $x \in \mathcal{X}_t(R_t)$. The optimization problem is an integer program, and as discussed in the previous section, for applications in transportation and logistics, it often exhibits network or near-network structure. This facilitates use of available software for integer programming. Though there are no theoretical guarantees when the problem does not exhibit pure network structure, this approach appears to work well in practice.

For each attribute $a \in \mathcal{A}$ and post-decision state R_t^x, Step 3 estimates how availability of an additional unit of a resource with attribute a would have impacted the prediction of future contribution. The difference is denoted by d_{ta}. Step 4 is a stochastic approximation iteration that

tunes the gradient of the approximation $\bar{V}^x(R_t^x, v_t)$ with respect to R_t^x to be closer to the estimated gradient d_t.

In the case of a separable concave approximation, an extra step after the tuning projects the approximation back into the space of concave functions, if the tuning step destroys concavity. Note that we cannot even allow intermediate functional approximations to be nonconcave because of the complexities this would introduce to the problem of optimizing over $x \in \mathcal{X}_t(R_t)$. For a discussion of methods for maintaining concavity, see Godfrey & Powell (2001), Topaloglu & Powell (2003*a*) and Powell et al. (2002*a*).

We mentioned earlier that reformulation of the dynamic programming recursion in terms of a post-decision state could be critical to the use of simulation-based methods. Let us now discuss how this relates to the above algorithm. One could imagine developing an analogous algorithm to approximate the pre-decision value function $V_t(R_t)$ using an approximation architecture $\bar{V}(R_t, v_t)$. Again, we would generate a random sequence, simulate the corresponding state trajectory, estimate gradients of predicted contribution, and update the approximation architecture accordingly. However, a difficulty arises in the Step 3, as we now explain.

In Step 3, q_t can be viewed as a sample estimate of

$$E_t \left[\max_{x \in \mathcal{X}_{t+1}(R_t^x + \hat{R}(R_t^x, \omega_{t+1}))} \left(C_{t+1} x + \bar{V}^x(\Delta_{t+1} x, v_t) \right) \right]. \tag{11}$$

It is easy to see that q_t is an unbiased estimate of this expectation, and this is important, since the stochastic approximation updates essentially average these estimates to better estimate the expectation.

Suppose now that we wish to adapt the algorithm so that it works with a pre-decision value function $\bar{V}(R_t, v_t)$. Then, in Step 3, we would want to generate an unbiased estimate of

$$\max_{x \in \mathcal{X}_t(R_t)} \left(C_t x + E_t \left[\bar{V}(R_t + \Delta_t x + \hat{R}(R_t + \Delta_t x, \omega_{t+1}), v_t) \right] \right), \tag{12}$$

based on a sampled value of ω_{t+1}. In general, this is not possible. Unlike the expectation of the maximum (11), there is no simple way of generating an unbiased estimate of the maximum of an expectation (12) based on a single sample of ω_{t+1}. An exception to this, however, arises in the case of a linear approximation architecture if $\hat{R}$ depends on R_t instead of R_t^x. In this event, we have

$$\begin{aligned} &\max\nolimits_{x \in \mathcal{X}_t(R_t)} \left(C_t x + E_t \left[\bar{V} \left(\Delta_t x + \hat{R}(R_t, \omega_{t+1}), v_t \right) \right] \right) \\ &\quad = \max\nolimits_{x \in \mathcal{X}_t(R_t)} \left(C_t x + \bar{V}(\Delta_t x, v_t) \right) + E_t \left[\bar{V} \left(\hat{R}(R_t, \omega_{t+1}), v_t \right) \right]. \end{aligned}$$

Clearly, an unbiased estimate can be generated by maximizing the first term on the right hand side and adding $\bar{V}(\hat{R}(R_t, \omega_{t+1}), v_t)$ generated from a single sample ω_{t+1}. On the other hand, when the approximation architecture is nonlinear or $\hat{R}$ depends R_t^x, an unbiased estimate can no longer be generated based on a single sample.

5 Mathematical programming

The problem formulation and algorithms we have presented evolved from a line of work on solution methods for large-scale problems in transportation and logistics. As of this writing, this field is dominated by algorithms developed within the mathematical programming community for handling high-dimensional resource allocation problems. We briefly review a simple deterministic model (for which there is a vast array of algorithms) and then introduce two competing themes that have emerged from within this community for introducing uncertainty.

5.1 Deterministic mathematical programming

In the field of transportation and logistics, the most common modelling and algorithmic strategy is to formulate the problem deterministically and then use classical math programming algorithms to solve the resulting model. For example, consider a deterministic version of the problem we introduced in section 2.1 resulting from setting $\hat{R}(R_t^x, \omega_{t+1}) = \hat{R}_{t+1}$ for a deterministic sequence $\hat{R}_1, \ldots, \hat{R}_T$. This would lead to a deterministic optimization problem:

$$\max_{x_0,\ldots,x_{T-1}} \sum_{t=0}^{T-1} C_t x_t \tag{13}$$

subject to, for $t = 0, \ldots, T-1$:

$$\begin{aligned} A_t x_t &= R_t && (14)\\ U_t x_t &\leq u_t && (15)\\ R_{t+1} &= \Delta_t x_t + \hat{R}_{t+1} && (16)\\ x_t &\geq 0 \text{ and integer-valued} && (17) \end{aligned}$$

As with our stochastic formulation from section 2.1, Equation (14) typically represents flow conservation. Equations (15) and (17) impose upper and lower bounds on the flow. Equation (16) captures the state dynamics.

Deterministic problems like the one we have formulated often result in large, time staged linear integer programs. These can sometimes be solved effectively. However, they completely ignore uncertainty and some of the more complex dynamics that can arise in real-world settings.

5.2 Stochastic Programming

The most common approach for modelling resource allocation problems in practice is to formulate the problem deterministically and apply mathematical programming algorithms developed for this setting. When uncertainty is introduced into mathematical programming, we find ourselves in a subfield called *stochastic programming*. In a sense, Markov decision processes might be viewed as an extension of stochastic systems models to incorporate optimization, while stochastic programming might be viewed as an extension of (deterministic) mathematical programming to incorporate uncertainty.

Two algorithmic strategies have emerged from the stochastic programming community. The first, covered in section 5.2.1, uses an explicit representation of decisions in the future for each of a finite set of scenarios. This approach is fundamentally different approach from dynamic programming. The second strategy, called Benders decomposition (section 5.2.2) uses a method for approximating the downstream impact of decisions. This is much closer in spirit to dynamic programming.

5.2.1 Scenario methods

The oldest strategy to solving linear programs under uncertainty is stochastic programming using scenario methods. The roots of this approach date to the work by Dantzig (Dantzig & Ferguson (1956)), but the earliest serious investigation of stochastic programs were primarily due to Wets (Wets (1966*a*),Wets (1966*b*), Walkup & Wets (1967),Wets (1974)). For modern reviews of this field, see Birge & Louveaux (1997), Infanger (1994), Kall & Wallace (1994) and Sen & Higle (1999)).

As an example, let us formulate a scenario-based stochastic program to solve a problem similar to that posed by our model of section 2.1. Let uncertainty be represented in terms of a trajectory of random outcomes $\omega = (\omega_1, \ldots, \omega_T)$. Let Ω be the set of possible "scenarios" ω, and let $p(\omega)$ be the probability that each $\omega \in \Omega$ will occur. The random impact on resources at time t is assumed to take the form $\hat{R}_t(\omega)$. Using this representation of uncertainty, consider solving the following optimization problem:

$$\max_{x_0, \{x_t(\omega)|t=1,\ldots,T-1, \omega \in \Omega\}} C_0 x_0 + \sum_{\omega \in \hat{\Omega}} p(\omega) \sum_{t=1}^{T-1} C_t x_t(\omega) \tag{18}$$

subject to the following first stage constraints for $\omega \in \Omega$:

$$\begin{aligned} A_0 x_0 &= R_0 && (19)\\ U_0 x_0 &\leq u_0 && (20)\\ R_1(\omega) &= \Delta_0 x_0 + \hat{R}_1(\omega) && (21)\\ x_0 &\geq 0 \text{ and integer-valued} && (22) \end{aligned}$$

and the subsequent stage constraints, for $t = 1, \ldots, T-1$ and $\omega \in \Omega$:

$$\begin{aligned} A_t x_t(\omega) &= R_t(\omega) && (23)\\ U_t x_t(\omega) &\leq u_t && (24)\\ R_{t+1}(\omega) &= \Delta_t x_t(\omega) + \hat{R}_{t+1}(\omega) && (25)\\ x_t(\omega) &\geq 0 \text{ and integer-valued} && (26) \end{aligned}$$

This formulation allows us to make a different set of decisions for each outcome, which means we are allowing a decision to "see" into the future. To prevent this behavior, we incorporate an additional constraint for each $t = 1, \ldots, T-1$ and pair of scenarios $\omega, \omega' \in \Omega$ for which $(\omega_1, \ldots, \omega_t) = (\omega'_1, \ldots, \omega'_t)$:

$$x_t(\omega) = x_t(\omega') \tag{27}$$

The constraints (27) are called *nonanticipativity constraints*.

We note that there is no explicit use of a state variable. This approach has proven useful in the financial services sector where random processes (e.g. interest rates, currency fluctuations, stock prices) can be correlated in a very complex way. This technique is widely used in financial asset allocation problems for designing low-risk portfolios (Mulvey & Vladimirou (1992), Mulvey (2001)), and a variety of specialized algorithms have been designed to help solve these problems (Lustig et al. (1991), Mulvey & Ruszczyński (1995), Mulvey & Ruszczyński (1991)).

The optimization problem represented by equations (18)-(27) requires determining all decisions over all scenarios (over all time periods) at the same time. Not surprisingly, the resulting optimization problem is *much* larger than its deterministic counterpart. In addition, the formulation typically destroys integrality properties that might exist in the original problem. The solution of stochastic integer programs is an emerging area of research (see Sen (to appear) for a thorough review of this field). This approach is generally not computationally tractable in the context of large scale resource allocation.

5.2.2 Benders Decomposition

An alternative strategy, which exploits the structure of linear programs, is to replace the second term on the right hand side of equation (18) with a series of cuts that captures the structure of this function. We discuss this approach in the context of a two-stage problem – that is, the case of $T = 2$. The idea is to iteratively solve a sequence of "master problems" of the form:

$$\max_{x_0,z} \quad C_0 x_0 + z \tag{28}$$

subject to the first stage constraints (19)-(22) and the constraints:

$$z - \beta_i x_0 \quad \leq \quad \alpha_i, \quad \forall \quad i = 1, ..., n \tag{29}$$

where β_i and α_i are generated by solving the dual of a second stage problem:

$$\max_{x_1(\omega)} C_1 x_1(\omega) \tag{30}$$

subject to constraints (23)-(26). The second stage is solved for a single $\omega \in \Omega$, so the problem is no larger than a one-period deterministic problem. After solving the second stage problem for a single scenario, the dual information is used to update the cuts in equation (29). There are different strategies for creating and updating cuts. The "L-shaped decomposition algorithm" (Van Slyke & Wets (1969)) solves two-stage problems with a finite set of scenarios. In one of the major breakthroughs in the field, Higle & Sen (1991) describe the "stochastic decomposition" algorithm that generalizes the L-shaped algorithm for problems with an infinite number of scenarios. Ruszczyński (1993) and Chen & Powell (1999) present variations of Benders for multistage problems. Figure 1 provides a sketch of the CUPPS algorithm in Chen & Powell (1999) which is basically a generalization of the L-shaped algorithm for multistage problems.

The appeal of Benders decomposition is seen by simply comparing the problem of solving equation (28) along with equation (30) to the challenge of solving equation (18). Benders decomposition

Step 1. Solve the following *master problem*:

$$x_0^n \in \arg\max\{C_0 x + z : A_0 x_0 = R_0,\ z - \beta_k x \leq \alpha_k,\ k = 1, \ldots, n-1,\ x \geq 0\}$$

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

$$v(x_0^n, \omega^n) \in \arg\min\{(\hat{R}_1(\omega^n) + \Delta_0 x_0^n)^T v : A_1^T v \geq C_1\}$$

Augment the set of dual vertices by:

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

Step 3. Set:

$$v^n(\omega) \in \arg\min\{(\hat{R}_1(\omega) + \Delta_0 x_0^n) v : v \in \mathcal{V}^n\} \text{ for all } \omega \in \Omega$$

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

$$\alpha_n + \beta_n x_0 \equiv \sum_{\omega \in \Omega} p(\omega) \left(\hat{R}_1(\omega) + \Delta_0 x_0\right)^T v^n(\omega)$$

Figure 1: Sketch of the CUPPS algorithm

exploits the structure of a linear program, however the rate of convergence remains an open question. Powell et al. (2002*a*), for example, found that Benders algorithm could require hundreds of iterations before it outperformed a simple deterministic approximation even for relatively small problems (the rate of convergence slows as the problem size increases).

Benders can be used for multistage problems by simply stepping forward through time, a technique that is sometimes referred to as nested Benders decomposition (see Pereira & Pinto (1991) for an application in an energy setting and Archibald et al. (1999) for an application to reservoir optimization). The basic strategy is identical to that used in approximate dynamic programming. For an analysis of the convergence of these techniques see Ruszczyński (1993) and Chen & Powell (1999).

6 Approximate Dynamic Programming

The algorithms we have presented emerged from a thread of research that grew to a large extent independently from the broader approximate dynamic programming literature – which includes, for example, work under the guise of "neuro-dynamic programming," "reinforcement learning," and "heuristic dynamic programming." We refer the reader to Bertsekas & Tsitsiklis (1996), Sutton & Barto (1998), and Van Roy (2001) for surveys of this literature. There are a number of ideas in common between the algorithm we have presented and those proposed in this approximate dynamic programming literature. In this section, we discuss these similarities and also some notable differences.

Like algorithms from the broader approximate dynamic literature, the algorithm presented in section 4 approximates a dynamic programming value function for use in decision-making. Another striking commonality is the use of simulated state trajectories and stochastic approximation updates. In this respect, the algorithm of section 4 is closely related to temporal-difference learning (Sutton (1988)), Q-learning (Watkins (1989) and Watkins & Dayan (1992)), and SARSA (Rummery & Niranjan (1994)). Some work in the reinforcement learning literature highlights the importance

of simulated state trajectories towards reducing approximation error (Sutton (1996) and Tsitsiklis & Van Roy (1997)). Perhaps this phenomenon has also contributed to the empirical success in transportation of algorithms like that of section 4.

The formulation of dynamic programs around the pre-decision state is the most standard treatment in textbooks, though even Bellman's original treatment of dynamic programming (Bellman (1957), p. 142) included an example of a dynamic programming recursion in terms of a post-decision state. In our context, such a reformulation of the dynamic programming recursion becomes necessary to enable computation of unbiased estimates of the right hand side via simulation. Such a reformulation is employed for the same reason by Q-learning and SARSA. One interesting difference, though, is that the post-decision state used by Q-learning and SARSA is taken to be the state-action pair – in our context, this would be (R_t, x_t). The algorithm we have presented, on the other hand, makes use of a more parsimonious representation $R_t^x = R_t + \Delta_t x_t$ of the post-decision state. This is possible because of special structure associated with our problem formulation, which makes R_t^x a sufficient statistic for predicting future contribution. Since Q-learning is designed for more general Markov decision processes that do not exhibit such structure, it can not make use of such a sufficient statistic. The idea of using a parsimonious sufficient statistic has also been proposed in an application of approximate dynamic programming to inventory management (Van Roy et al. (1997)). The use of the post-decision state variable is implicit in the work of Powell (1988) and Cheung & Powell (1996) for a stochastic fleet management problem, and is explicit in Godfrey & Powell (2002) for the same problem class, and Papadaki & Powell (2002) and Spivey & Powell (to appear) in other problem classes.

Another significant difference in the algorithm of section 4 is that the updates adapt the derivative of the value function with respect to state, whereas most algorithms in the approximate dynamic programming literature adapt the values themselves. Clearly, it is the derivative of the value function that matters when it comes to decision-making, since the value function is only used to compare relative values among states. An interesting issue to explore may be whether there are fundamental advantages to adapting derivatives rather than values. Though the idea of adapting derivatives has not been a focus of the approximate dynamic programming literature, it has received some attention. For example, this idea has been promoted by Werbos, who proposed a number of algorithms with this objective in mind (Werbos (1992*a*) and Werbos (1992*b*)).

An important departure from the broader approximate dynamic programming literature is in the use of integer programming methods to solve the decision optimization problem

$$\max_{x \in \mathcal{X}_t(R_t)} \left(C_{t+1} x + \bar{V}^x(\Delta_{t+1} x, v_{t+1}) \right).$$

This approach has been successfully applied to problems involving thousands of decision variables. Though there is some work in the approximate dynamic programming literature on approaches for dealing with high-dimensional decision spaces (see, e.g., Bertsekas & Tsitsiklis (1996), Crites & Barto (1994), and Guestrin et al. (2003)), to our knowledge, no other applications of approximate dynamic programming have dealt with such large numbers of decision variables.

The use of separable linear and concave approximations facilitate the use of integer programming methods to optimize decisions. Though separable approximators have been used regularly in the approximate dynamic programming literature (see, e.g., Sutton (1996)), there has not been a practice of restricting to linear or concave functions, and their use has not been motivated by a need to structure the decision optimization problem. The separable linear or concave structure is important – if more general nonlinear approximations were used this would likely destroy integrality

properties of underlying linear programs. The result would be a decision optimization problem that could not be handled effectively by integer programming methods.

Finally, it is worth noting that the scale of problems tackled by the first author and collaborators far exceeds most other applications of approximate dynamic programming reported in the literature. In particular, there are typically thousands of state variables and thousands of decision variables. For problems of such enormous scale, it becomes important to use methods for which compute time and memory requirements grow slowly with the number of state variables. In this respect, separable approximation architectures are advantageous because the number of parameters involved grows linearly in the number of state variables.

7 Experimental Comparisons

The techniques described in this chapter have evolved in the context of solving an array of very large-scale resource allocation problems. One of the challenges that we always face in the field of approximate dynamic programming is evaluating how well our techniques work. Bounds can be useful, but tend to be quite loose in practice.

For our problem class, we can report on two types of experiments. The first focuses on two-stage problems (make a decision in the first stage, see information in the second stage, make a second decision, stop), for which computationally tractable, provably convergent algorithms already exist using the principle of Benders decomposition (which we reviewed in section 5.2.2). The second set of experiments look at multiperiod problems, and compare against rolling horizon approximations that use deterministic forecasts of the future.

Figure 2 shows comparisons of three variations of Benders against a variation of the concave, separable approximation method (described in section 4.3) called the SPAR algorithm (see Powell et al. (2002*a*)). The algorithms were run on four sets of problem of increasing size (10, 25, 50 and 100 locations). The execution times are comparable (although the "L-shaped" algorithm is much slower than the others). But the results suggest that Benders exhibits higher errors after the same number of iterations, and that the error increases dramatically with problem size, whereas the SPAR algorithm actually improves. This is capturing the observation that in practice, large problems are reasonably approximated using separable approximations.

Benders decomposition can be run on multiperiod problems as well, but the rate of convergence appears to become prohibitively slow. The best competition comes from the standard engineering practice of solving sequences of deterministic approximations using a rolling horizon procedure. Topaloglu & Powell (2003*b*) reports on comparisons using linear and piecewise linear, concave value function approximations against a deterministic, rolling horizon procedure. Table 1 summarizes one set of experiments that considered problems with 10, 20 and 40 locations. Each cell in this table gives the average objective function for each method divided by the average of all the posterior bounds obtained by calculating the (outcome-dependent) optimal solutions for each of the 100 sample realizations tested. This set of experiments focused on an application with different equipment types (such as box cars) which can be described purely based on the type of equipment and their location. It is possible to substitute different equipment types for the same order (at a cost), as well as substituting equipment of the same type from different locations. Separable concave value functions are estimated for each location and equipment type, which would suggest a significant potential for error. Just the same, separable concave approximations work quite well

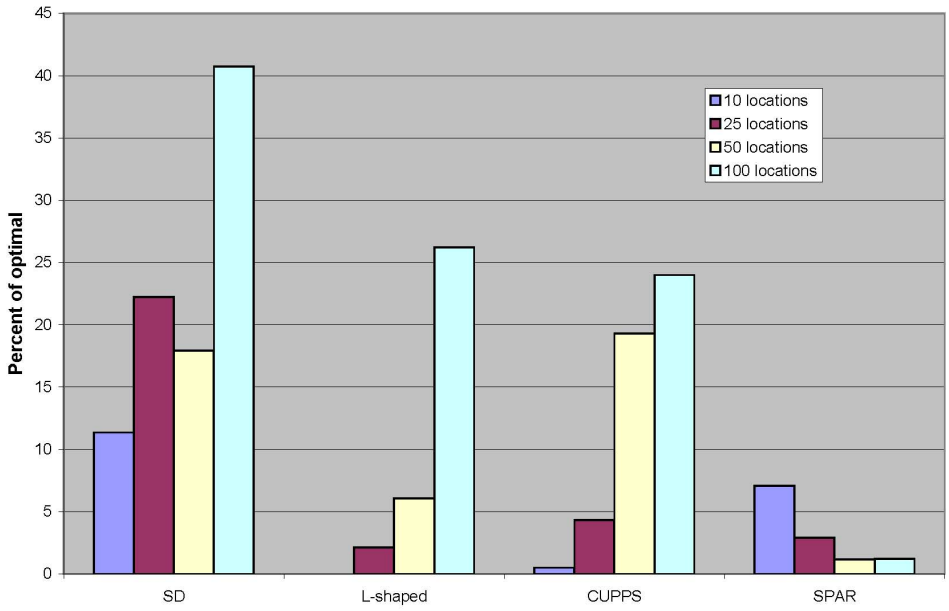

Figure 2: The quality of the solution produced by variations of Benders and separable, concave approximations (SPAR) expressed as a percent of the optimal solution found using L-shaped decomposition. Based on results from Powell et al. (2002*a*).

Locations	Approx.	Mean	Std. dev.	Percentiles 5^{th}	50^{th}	95^{th}
10	Linear	84.55	2.339	80.94	84.45	88.43
	Separable Concave	95.18	2.409	91.55	95.46	99.38
	Rolling Horizon	91.45	2.348	87.96	91.97	95.74
20	Linear	80.52	2.463	76.91	80.45	85.25
	Separable Concave	95.48	2.153	91.91	95.33	98.96
	Rolling Horizon	88.91	1.930	85.68	88.52	91.91
40	Linear	74.13	1.816	72.02	74.14	77.56
	Separable Concave	92.21	0.465	91.55	92.89	93.22
	Rolling Horizon	86.89	0.772	85.08	86.77	87.11

Table 1: Comparisons of a deterministic rolling horizon procedure against value function approximations using linear and separable concave architectures, from Topaloglu & Powell (2003*b*). Each cell gives the average objective function value normalized by the average of the posterior bounds for each sample realization.

against the linear approximations (which works the worst) and the rolling horizon procedure.

As of this writing, we do not know exactly how well these methods work on general, multistage problems given the lack of effective, computationally tractable competing algorithms. There are many unanswered questions, but there has been enough progress to suggest that this is a promising line of investigation for this problem class.

8 Conclusion

Dynamic resource allocation problems typically lead to very high dimensional stochastic dynamic programs. We have presented approximation algorithms representative of some that have been successful in applications involving transportation and logistics. The algorithms incorporate several key ideas that address the three curses of dimensionality. Among these ideas are a reformulation of the dynamic programming recursion around a post-decision state variable, simulation-based stochastic approximation techniques, and continuous approximation architectures with functional forms that facilitate integer programming solutions to decision optimization sub-problems.

The approach has worked well for problems with thousands of discrete resources with tens of thousands of different attributes (in effect, a state vector with tens of thousands of dimensions). Furthermore, the rate of convergence of the learning of the value function approximations appears to be reasonably fast. In some applications, we found that the learning stabilizes within 20 to 40 iterations. This is important, since large problems can require as much as an hour per iteration.

Acknowledgements

This research of the first author was supported in part by grant AFOSR-F49620-93-1-0098 from the Air Force Office of Scientific Research. The second author was supported in part by the NSF under CAREER Grant ECS-9985229 and by the ONR under grant MURI N00014-00-1-0637.

References

Alden, J. & Smith, R. (1992), 'Rolling horizon procedures in nonhomogeneous Markov decision processes', *Operations Research* **40**(Supp. No. 2), S183–S194. 6

Archibald, T. W., Buchanan, C. S., McKinnon, K. I. M. & Thomas, L. C. (1999), 'Nested Benders decomposition and dynamic programming for reservoir optimisation', *J. Oper. Res. Soc.* **50**, 468–479. 14

Bellman, R. (1957), *Dynamic Programming*, Princeton University Press, Princeton. 15

Bertsekas, D. & Tsitsiklis, J. (1996), *Neuro-Dynamic Programming*, Athena Scientific, Belmont, MA. 14, 15

Birge, J. & Louveaux, F. (1997), *Introduction to Stochastic Programming*, Springer-Verlag, New York. 12

Chen, Z.-L. & Powell, W. (1999), 'A convergent cutting-plane and partial-sampling algorithm for multistage linear programs with recourse', *Journal of Optimization Theory and Applications* **103**(3), 497–524. 13, 14

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. 15

Crites, R. & Barto, A. (1994), Elevator group control using multiple reinforcement learning agents, Technical report. 15

Dantzig, G. & Ferguson, A. (1956), 'The allocation of aircrafts to routes: An example of linear programming under uncertain demand', *Management Science* **3**, 45–73. 12

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. 8, 15

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. 10

Guestrin, C., Koller, D. & Parr, R. (2003), Efficient solution algorithms for factored MDPs, Technical report. 15

Higle, J. & Sen, S. (1991), 'Stochastic decomposition: An algorithm for two stage linear programs with recourse', *Mathematics of Operations Research* **16**(3), 650–669. 13

Infanger, G. (1994), *Planning under Uncertainty: Solving Large-scale Stochastic Linear Programs*, The Scientific Press Series, Boyd & Fraser, New York. 12

Kall, P. & Wallace, S. (1994), *Stochastic Programming*, John Wiley and Sons, New York. 12

Kleywegt, A. & Papastavrou, J. (1998), 'Acceptance and dispatching policies for a distribution problem', *Transporation Science* **32**(2), 127–141. 6

Lustig, I., Mulvey, J. & Carpenter, T. (1991), 'Formulating stochastic programs for interior point methods', *Operations Research* **39**, 757–770. 13

Mulvey, J. (2001), 'Introduction to financial optimization:Mathematical Programming special issue', *Mathematical Programming: Series B* **89**, 205–216. 13

Mulvey, J. M. & Ruszczyński, A. (1991), 'A diagonal quadratic approximation method for large scale linear programs', *Operations Research Letters* **12**, 205–215. 13

Mulvey, J. M. & Ruszczyński, A. J. (1995), 'A new scenario decomposition method for large-scale stochastic optimization', *Operations Research* **43**(3), 477–490. 13

Mulvey, J. M. & Vladimirou, H. (1992), 'Stochastic network programming for financial planning problems', *Management Science* **38**(8), 1642–1664. 13

Papadaki, K. & Powell, W. B. (2002), 'A monotone adaptive dynamic programming algorithm for a stochastic batch service problem', *European Journal of Operational Research* **142**(1), 108–127. 15

Pereira, M. & Pinto, L. (1991), 'Multistage stochastic optimization applied to energy planning', *Mathematical Programming* **52**, 359–375. 14

Powell, W. B. (1988), A comparative review of alternative algorithms for the dynamic vehicle allocation problem, *in* B. Golden & A. Assad, eds, 'Vehicle Routing: Methods and Studies', North Holland, Amsterdam, pp. 249–292. 15

Powell, W. B. (1996), 'A stochastic formulation of the dynamic assignment problem, with an application to truckload motor carriers', *Transportation Science* **30**(3), 195–219. 2

Powell, W. B. & Topaloglu, H. (2003), Stochastic programming in transportation and logistics, *in* A. Ruszczyński & A. Shapiro, eds, '*Handbook in Operations Research and Management Science*, Volume on *Stochastic Programming*', Elsevier, Amsterdam, pp. 555–635. 8

Powell, W. B., Ruszczyński, A. & Topaloglu, H. (2002*a*), Learning algorithms for separable approximations of stochastic optimization problems, Technical report, Princeton University, Department of Operations Research and Financial Engineering. 10, 14, 16, 17

Powell, W. B., Shapiro, J. A. & Simão, H. P. (2002*b*), 'An adaptive dynamic programming algorithm for the heterogeneous resource allocation problem', *Transportation Science* **36**(2), 231–249. 2, 8

Rummery, G. & Niranjan, M. (1994), On-line Q-learning using connectionist systems, Technical report, Cambridge University Engineering Department Technical Report CUED/F-INFENG/TR166. 14

Ruszczyński, A. (1993), 'Parallel decomposition of multistage stochastic programming problems', *Math. Programming* **58**(2), 201–228. 13, 14

Sen, S. (to appear), Algorithms for stochastic, mixed-integer programs, *in* G. L. Nemhauser, ed., '*Handbook in Operations Research and Management Science*, Volume on *Discrete Optimization*', North Holland, Amsterdam. 13

Sen, S. & Higle, J. (1999), 'An introductory tutorial on stochastic linear programming models', *Interfaces* **29**(2), 33–61. 12

Spivey, M. & Powell, W. B. (to appear), 'The dynamic assignment problem', *Transportation Science*. 8, 15

Sutton, R. (1988), 'Learning to predict by the methods of temporal differences', *Machine Learning* **3**, 9–44. 14

Sutton, R. & Barto, A. (1998), *Reinforcement Learning*, The MIT Press, Cambridge, Massachusetts. 14

Sutton, R. S. (1996), Generalization in reinforcement learning: Successful examples using sparse coarse coding, *in* M. E. H. D. S. Touretzky, M. C. Mozer, ed., 'Advances in Neural Information Processing Systems', Vol. 19, pp. 1038–1044. 15

Topaloglu, H. & Powell, W. B. (2003*a*), 'An algorithm for approximating piecewise linear concave functions from sample gradients', *Operations Research Letters* **31**(1), 66–76. 10

Topaloglu, H. & Powell, W. B. (2003*b*), Dynamic programming approximations for stochastic, time-staged integer multicommodity flow problems, Technical report, Cornell University, Department of Operations Research and Industrial Engineering. 16, 17

Tsitsiklis, J. & Van Roy, B. (1997), 'An analysis of temporal-difference learning with function approximation', *IEEE Transactions on Automatic Control* **42**, 674–690. 15

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. 14

Van Roy, B., Bertsekas, D. P., Lee, Y. & Tsitsiklis, J. N. (1997), A neuro-dynamic programming approach to retailer inventory management, *in* 'Proceedings of the IEEE Conference on Decision and Control'. 15

Van Slyke, R. & Wets, R. (1969), 'L-shaped linear programs with applications to optimal control and stochastic programming', *SIAM Journal of Applied Mathematics* **17**(4), 638–663. 13

Walkup, D. & Wets, R. (1967), 'Stochastic programs with recourse', *SIAM Journal of Applied Mathematics* **15**, 1299–1314. 12

Watkins, C. (1989), Learning from delayed rewards, Ph.d. thesis, Cambridge University, Cambridge, UK. 14

Watkins, C. & Dayan, P. (1992), 'Q-learning', *Machine Learning* **8**, 279–292. 14

Werbos, P. J. (1992*a*), Approximate dynamic programming for real-time control and neural modelling, *in* D. A. White & D. A. Sofge, eds, 'Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches'. 15

Werbos, P. J. (1992*b*), Neurocontrol and supervised learning: an overview and valuation, *in* D. A. White & D. A. Sofge, eds, 'Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches'. 15

Wets, R. (1966*a*), 'Programming under uncertainty: The equivalent convex program', *SIAM Journal of Applied Mathematics* **14**, 89–105. 12

Wets, R. (1966*b*), 'Programming under uncertainty: The solution set', *SIAM Journal of Applied Mathematics* **14**, 1143–1151. 12

Wets, R. (1974), 'Stochastic programs with fixed recourse: The equivalent deterministic problem', *SIAM Review* **16**, 309–339. 12