Proceedings of the 2005 Winter Simulation Conference
M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds.

THE OPTIMIZING-SIMULATOR: MERGING SIMULATION AND OPTIMIZATION USING APPROXIMATE DYNAMIC PROGRAMMING

Warren B. Powell

Department of Operations Research and Financial Engineering
Princeton University
Princeton, NJ 08544, U.S.A.

ABSTRACT

There has long been a competition between simulation and optimization in the modeling of problems in transportation and logistics, machine scheduling and similar high-dimensional problems in operations research. Simulation strives to model operations, often using rule-based logic. Optimization strives to find the best possible solution, minimizing costs or maximizing profits. In this tutorial, we show how these two modeling technologies can be brought together, combining the flexibility of simulation with the intelligence of optimization.

1 INTRODUCTION

There is a broad range of problems that arise under the umbrella of dynamic resource allocation in which there is a healthy debate: should we optimize or simulate? This question has been particularly prominent in the transportation and logistics community where optimization has been the favored strategy of academics, while many practitioners prefer simulation. In this paper, we propose that these problems can be best modeled using a hybrid that we refer to as the "optimizing-simulator."

A nice illustration of the debate is the modeling of military airlift operations. The analysis group at the Airlift Mobility Command (AMC), which plans the movements of cargo aircraft for the U.S. air force, has long used simulation, originally in a model called "MASS" and more recently in a rewritten model called "AMOS." The model simulates the movement of a variety of cargo aircraft (e.g. C-5, C-17, C-141, Boeing 747, KC-10 and KC-135) to move "requirements" that consist of a mixture of freight and passengers. The demands on the system are captured using a "TPFDD" file (time-phased, force deployment dataset) that specifies what freight and passengers have to be moved, including origin and destination, time of availability and the characteristics of the freight. Schank et al. 1991 provides a review of simulation models for military airlift.

Competing against AMOS is a series of optimization-based models such as CONOP (Killingsworth and Melody 1997), THRUPUT II (Rosenthal et al. 1997), and most notably NRMO (Baker et al. 2002). These models represent the movement of aircraft and requirements (NRMO also models pilots at a simple level) as a linear (integer) program which is solved using a commercial solver such as Cplex. The most common criticism of these models by practitioners is that they assume too much about what is known when making a decision (decisions at time t "see" information about the future that would not occur in practice). In addition, these models are not able to handle any of the many sources of uncertainty (last minute customer requests, weather delays, equipment failures). Often unstated but just as important is the difficulty of handling complex system dynamics (for example, an aircraft failure may block a runway at a small airbase if the refueling station is located near the runway).

The issues that arise in the modeling of military airlift also arise in a variety of problems in transportation and logistics, supply chain management and even certain types of manufacturing operations such as flexible flow shops. The problem is that we are managing multiple resources which have to be assigned to various tasks where we have flexibility in the assignment of which resource should be assigned to each task. These are often referred to as dynamic assignment problems, where we have to assign resources to tasks over time in the presence of different types of uncertainty. We have to decide which resource is best for each task, recognizing that serving a task not only occupies the resource for a period of time, but also changes the attributes of the resource (an aircraft moving a load of freight, a machine that has to incur a setup to handle a particular job).

The case for using simulation is that it easily handles uncertainty and the modeling of complex system dynamics. A problem that simulation modelers face is the dimensionality of the decision vectors. Most simulation models either do not require simulating decisions (when a job finishes

machine i, it goes to machine j with probability p_{ij}) or use fairly simple rules (when the job finishes machine i, assign it to machine j_1 if the queue is less than n, otherwise assign it to machine j_2, ...). By contrast, an optimization model can formulate the problem of assigning resources to tasks and solve it as a linear program. However, standard practice (this is especially true in the transportation and logistics community) is to then formulate the entire problem (over time) as a single, large optimization problem. In between are simulation models that solve a sequence of smaller optimization problems at each point in time. These models typically ignore the impact of decisions now on the future.

Often overlooked in the debate is the property that the behavior of a simulation is guided by user-specified rules, while the behavior of an optimization model is determined by a cost function (as well as the constraints). Each approach brings specific strengths. If the behavior of a simulation model is judged (typically by a domain expert) to be incorrect, it can be modified by changing the relevant rules. By contrast, tuning an optimization model requires changing the costs and hoping that the resulting change produces the desired behavior. Designing a cost model which produces the desired behavior can be difficult and time-consuming.

In this paper, we argue that optimization and simulation can be merged, producing a technology that handles the randomness and complexities of simulation with the intelligence of optimization. Unlike simulations that solve sequences of myopic optimization models, we propose a strategy that optimizes over time, and show that for some problem classes, the result is solutions that can compete with an optimization problem. At the same time, we wish to retain at least some of the ability possessed by simulation models where we can modify the behavior of the system by changing rules, rather than by explicitly tuning the costs.

Our presentation begins in section 2 with an introduction to the military airlift problem which has seen considerable attention from the simulation and optimization communities. We use this setting to introduce a mathematical model that forms the basis for the rest of the presentation. Section 3 then presents two types of myopic policies: one that uses rule-based logic, while the other uses a cost-based objective function. Then, section 4 describes how to use approximate dynamic programming to produce decision functions that optimize over time (rather than just at a point in time). This logic takes us from a classical simulation framework to one that adaptively learns better behaviors. This logic can work quite well as long as the cost function is accurate, but a common problem is that even optimal solutions may be unacceptable. Section 5 shows how a cost-based model can be combined with rule-based logic (expressed in the form of fairly simple rules known as low-dimensional patterns).

2 AN ASSET ALLOCATION PROBLEM

The ideas in this paper are designed for modeling a broad range of complex resource allocation problems. At the same time, we recognize that a specific example can greatly facilitate communication. For this reason, we briefly present the military airlift problem in section 2.1. We then present a general mathematical modeling framework for resource allocation in section 2.2.

2.1 A Military Airlift Problem

The military airlift problem involves modeling the movement of aircraft (possibly including pilots) around the world to move a set of cargo (which can include a mixture of passengers and freight), as illustrated in Figure 1. The cargo becomes available at a specified point in time, and the goal is to move it as quickly as possible.

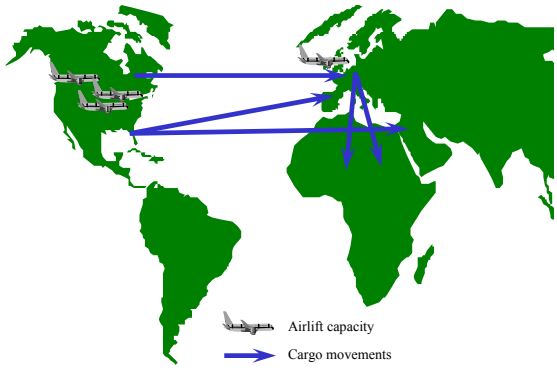


Figure 1: A military Airlift Problem

Modeling an airlift problem begins with the challenge of choosing an aircraft to move a block of cargo. Choosing an aircraft also implies choosing the route the aircraft will follow as it moves empty to pick up the cargo, and then loaded to get the cargo to the destination. Each aircraft has its own range before needing to refuel, and this range will depend on how much cargo is being carried. It is reasonable to specify different routes (sequences of airbases) through which the aircraft can move from origin to destination, but it is necessary to determine whether the movement is feasible. For example, it is often necessary to use fairly small airbases which can quickly reach capacity. The simulator has to determine if an airbase will be at capacity at the time the aircraft needs to move through the airbase. If an airbase is judged to be at capacity, a different route (or aircraft) will have to be used.

Once an aircraft is chosen to move a specific block of cargo, it is also necessary to simulate loading and unloading (which may need special equipment), refueling and repairs. A model may also capture the movement of crews, which

will require capturing the rules of what a crew can and cannot do.

An airlift simulation may also have to capture several sources of randomness. Requests to move cargo may be known in advance (but changes may be made at the last minute), or requests may arrive randomly over time. Aircraft may be delayed by weather or the need for repairs.

2.2 A Mathematical Model

In this section, we present a general mathematical model of a dynamic resource allocation problem, using from time to time the airlift mobility problem to illustrate the notation. We assume that our problem consists of assigning "resources" to "demands." We model resources (e.g. aircraft) using:

a = The vector of attributes that describe a single resource, $a = a_1, a_2, \ldots, a_n$.
$\mathcal{A}$ = The set of possible attributes.
R^A_{ta} = The number of aircraft (our resources) with attribute a at time t.
R^A_t = $(R^A_{ta})_{a\in\mathcal{A}}$.
$\hat{R}^A_{ta}(R_t)$ = The change in the number of resources with attribute a due to exogenous information that arrived during time interval t.
$\hat{R}^A_t$ = $(\hat{R}^A_{ta})_{a\in\mathcal{A}}$.

A given aircraft will have a vector of attributes a_t at time t. It is often the case that an aircraft moving from location i to location j will, at time t, be enroute, arriving at some point in the future. We model the *actionable time* as one of the attributes. The function $\hat{R}^A_{ta}(R_t)$ represents random information about aircraft. It could include new aircraft that unexpectedly enter the system ($\hat{R}^A_{ta}(R_t)$ would be the number of new aircraft with attribute a that are learned by time t). Alternatively, a failure or delay would be captured by subtracting an aircraft (with attribute a) from R_{ta} and adding it to $R^A_{ta'}$, if a' is the new attribute vector.

We assume that decisions are made in discrete time, but that exogenous information and physical events occur in continuous time. We label the time interval between $t-1$ and t as time interval t, and we assume that all exogenous information (such as $\hat{R}^A_t$) arrives continuously during time interval t. In the formal language of probability, we would let Ω be the set of elementary outcomes of all exogenous information, and let $\mathcal{F}_t$ be the sigma-algebra defined by the information available up through time t. In our notation, any variable indexed by t is therefore $\mathcal{F}_t$-measurable. Equivalently, if a variable is indexed by t, it can be computed using the information known up through time t.

We model demands in a similar way:

b = The vector of attributes that describe a single demand, $b = b_1, b_2, \ldots, b_n$.
$\mathcal{B}$ = The set of possible demand attributes.
R^D_{tb} = The number of demands with attribute b at time t.
R^D_t = $(R^D_{tb})_{b\in\mathcal{B}}$.
$\hat{R}^D_{tb}(R^D_t)$ = The change in the number of demands with attribute b due to exogenous information that arrived during time interval t.
$\hat{R}_t$ = $(\hat{R}^D_{tb})_{b\in\mathcal{B}}$.

Here, $\hat{R}^D_{tb}(R_t)$ will capture new customer requests and possibly changes in customer requests. Included in the vector b will be information about pickup and delivery time windows. R_{tb} might represent the size of the request (e.g. number of pounds of cargo), or simply the number of cargos (each of which require a whole aircraft) with a particular vector of characteristics.

Our notation allows us to write the state of the system (more precisely, the resource state) using:

$$R_t = (R^A_t, R^D_t).$$

If we wish to add pilots, we might introduce a vector R^P_t and add this to the resource state vector.

Decisions are modeled using:

d = A type of decision which can be used to act on a resource with attribute a.
$\mathcal{D}_a$ = The set of decisions which can be used to act on a resource with attribute a.
x_{tad} = The number of resources of type a that are acted on with a decision of type $d \in \mathcal{D}_a$.
x_t = $(x_{tad})_{a\in\mathcal{A},d\in\mathcal{D}_a}$
$\mathcal{X}_t$ = The feasible region for x_t.

For our presentation, we need to distinguish decisions that couple two resources (a job is assigned to a machine, an aircraft moves a load of freight) from decisions that simply modify a machine (the machine incurs a setup, the aircraft has to be repaired). We let $\mathcal{D}^D$ represent decisions to couple two resources (in our setting, assigning a resource to serve a demand) and we let $\mathcal{D}^M$ represent "modify" decisions, where $\mathcal{D} = \mathcal{D}^D \cup \mathcal{D}^M$.

At a minimum, the decision vector must be nonnegative and satisfy flow conservation:

$$\sum_{d\in\mathcal{D}} x_{tad} = R_{ta}. \quad (1)$$

The decision set $\mathcal{D}$ might include flying from one location to another, loading, unloading, repairing, refueling or simply sitting (the "do nothing" option). In principle we can include decisions that act on aircraft as well as the cargo (or pilots), but we assume for our presentation that we have a single *active layer* (the aircraft) and a single *passive layer* (the demands). If we included pilots, this would also be an active layer.

We model the effect of a decision on a resource using the *modify function*:

$$M(t, a, d) \rightarrow (a', c, \tau).$$

The modify function tells us that if we act on a resource with attribute a at time t with decision d then the result is a modified resource with attribute a' (e.g. the plane changes location or is repaired), generates a cost (or contribution) c, where τ is the time required to finish the action (τ can also be captured as an attribute). It is useful to define the *terminal attribute function* $a^M(t, a, d)$ which returns the attribute a'. For algebraic purposes, we also define the indicator function:

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

This notation allows us to write the resource dynamics using:

$$R^x_{t,a'} = \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}_a} \delta_{a'}(t, a, d) x_{tad} \quad (2)$$

$$R_{t+1,a'} = R^x_{t,a'} + \hat{R}_{t+1,a'}. \quad (3)$$

Here, R^x_t is called the *post-decision* resource vector, while R_t is the *pre-decision* resource vector. R^x_t captures what we know about the state of the resources after we have made a decision x_t, but before any new information has arrived. These two forms of the resource vector play important roles in the design of our algorithmic strategy. It is sometimes useful to represent these dynamics using a general transition function (vocabulary that is common in simulation but less familiar in optimization):

$$R_{t+1} = R^M(R_t, x_t, \hat{R}_{t+1}). \quad (4)$$

In this representation, x_t depends purely on R_t. Alternatively, we can write a recursion for the post-decision state variable using:

$$R^x_{t+1} = R^{M,x}(R^x_t, \hat{R}_{t+1}, x_{t+1}).$$

Here, x_{t+1} gets to "see" R^x_t as well as $\hat{R}_{t+1}$.

We next define the cost function using

$$\begin{aligned} c_{tad} &= \text{The cost of applying decision } d \text{ to a single resource with attribute vector } a \text{ at time } t. \\ C_t(R_t, x_t) &= \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}_a} c_{tad} x_{tad}. \end{aligned}$$

At this point we do not wish to describe how a decision is made (this is the focus of the remainder of the paper), but we define a family of decision functions (policies):

$$\begin{aligned} X^\pi_t(R_t) &= \text{A decision function which returns } x_t \in \mathcal{X}_t \text{ given a resource vector } R_t \text{ and decision rule } \pi \in \Pi. \\ \Pi &= \text{A set of decision functions (policies).} \end{aligned}$$

Our goal is to find the best decision function (or policy) to maximize:

$$\min_{\pi \in \Pi} \mathbb{E} \left\{ \sum_{t=0}^{T} \gamma^t C_t(R_t, X^\pi_t(R_t)) \right\}, \quad (5)$$

where γ is a discount factor.

At this point it is useful to discuss the cost function which was introduced with little discussion. In practice, many simulation models do not even have a cost function (this is true of the airlift simulator). Simulation models do not always need a cost function (decisions may be made using predefined rules). For problems in transportation and logistics, it is usually possible to define a basic cost function (e.g. total miles traveled), but for many complex problems, such cost functions do not capture all the issues.

In practice, finding an optimal solution to (5) is computationally intractable, and many modelers will claim that the goal is not to optimize anything, but rather to simulate. In the vast majority of our own work, we are also focusing on simulating an existing decision process, but we have found that real-world decision making tends to be rational (that is, it is approximately optimizing an appropriate cost function). We revisit this topic later in the paper.

3 SIMULATING MYOPIC POLICIES

We start by illustrating some simple policies that we can use if all we want to do is "simulate" the system. More than simply illustrating some basic ideas, these simple policies form the foundation of the more advanced policies we introduce later on.

3.1 A Rule-based Policy

Perhaps the most common policy used in simulation is a rule of the form "when in this state, take this action." This

is exactly what is used in the AMOS simulator used by the airlift mobility command. The rule works roughly as follows. At time t, there is a list of aircraft, sorted on the basis of availability, and a list of customer demands, also sorted based on the time at which they should leave. The problem is to find an aircraft that can satisfy the requirement.

The rule specifies that we take the first customer demand that needs to be served, and try to find an aircraft. This logic starts with the first available aircraft, and then undertakes a series of calculations to see if this assignment is feasible. For example, this feasibility check requires tracing the actual route the aircraft will use to be sure that airbase capacity constraints are not violated anywhere. If there is a violation, the aircraft is rejected and the logic moves to the next aircraft in the list. Note that the rule does not consider the distance from the current location of the aircraft to the origin of the order.

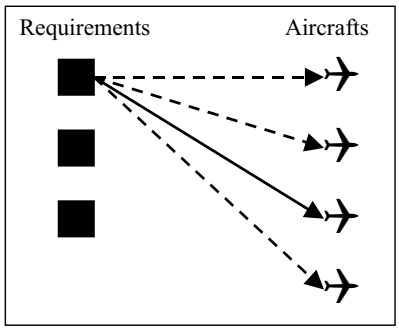
Rule-based policies have a difficult time making trade-offs (Should we use a larger aircraft that is farther away? Should we use an aircraft that is closer but is not yet available and will delay the movement of the order?). They also struggle with goals such as "We prefer not to use C-141's on movements into Saudi Arabia." But they offer tremendous flexibility, providing the analyst with direct control over the behavior of the system.

Rule-based policies have long been used as a practical tool for a variety of dynamic problems. In machine scheduling, shortest-job-first has long been recognized as an effective policy for minimizing the average wait time. In certain inventory problems, a myopic policy that specifies that you place an order $x_t = S - R_t$ whenever the inventory R_t falls below a specified level s can be optimal. These simple policies can be optimal, but typically only under very special situations.

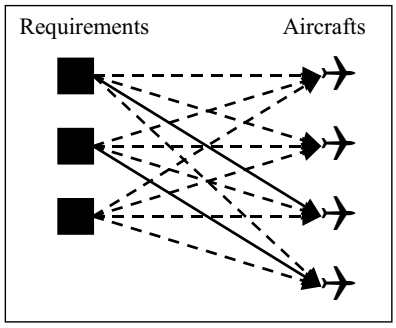
3.2 Myopic Cost-based Policies

Assume that we can measure the cost of assigning an aircraft with attribute a to a demand with attribute b. We might consider a single demand and a list of aircraft, and choose the aircraft with the lowest cost (Figure 2a). This seems like a trivial type of optimization, but it involves the transition from a rule-based to a cost-based policy. Instead of controlling the behavior of the system through a rule, we now face the challenge of designing a cost function that produces the desired behavior.

There is another "cost" to using a cost-based policy. In our original rule-based policy, we pick a demand, then pick an aircraft and then determine whether the assignment is feasible. If so, we are done. Determining feasibility (for the military airlift problem) is relatively time-consuming. With a cost-based policy, we first evaluate a series of possible assignments and then use an optimization algorithm



2a: One demand, multiple aircraft



2b: Multiple demands and aircraft

Figure 2: A Cost-based Policy for One Demand and Multiple Aircraft, or Multiple Demands and Aircraft Simultaneously.

to choose the best. We need to consider the time required to cost out all the possible assignments.

If we are going to generate costs for different assignments, then we might as well consider a set of aircraft and demands. We might restrict our set to aircraft and demands that can be acted on now (for example, the aircraft is sitting at an airbase waiting to be assigned, and the demand is available to be moved now). Or, we could also include aircraft and demands that are actionable within a specified planning horizon. We model this by defining:

$\mathcal{A}_t^{ph}$ = The subset of attributes where the resource (aircraft) is actionable within a specified planning horizon of time t.

$\mathcal{D}_t^{D,ph}$ = The set of decisions to assign aircraft to a demand that is actionable within a specified planning horizon of time t.

Each decision $d \in \mathcal{D}_t^{D,ph}$ corresponds to a decision to assign an aircraft to some type of demand. We designate the type of demand as b_d, since each element of $\mathcal{D}_t^{D,ph}$ has to correspond to an element of the demand attribute space $\mathcal{B}$. The problem of assigning multiple aircraft (more specifically, multiple types of aircraft) to multiple demands

(depicted in Figure 2b) is given by the linear program:

$$\max_{x_t \in \mathcal{X}_t} \sum_{a \in \mathcal{A}_t^{ph}} \sum_{d \in \mathcal{D}^{D,ph}} c_{tad} x_{tad} \tag{6}$$

subject to:

$$\sum_{d \in \mathcal{D}_t^{ph}} x_{tad} = R_{ta}^{A} \quad a \in \mathcal{A}_t^{ph} \tag{7}$$

$$\sum_{a \in \mathcal{A}_t^{ph}} x_{tad} = R_{tb_d}^{D} \quad d \in \mathcal{D}_t^{D,ph} \tag{8}$$

$$x_{tad} \geq 0. \tag{9}$$

Problem (6)-(9) is a fairly simple linear program. In fact, it is a network model and will return integer solutions. Problems with thousands of resource and demand types, if the linear program is properly engineered, can be solved on a state-of-the-art PC.

The myopic problem represents a class of decision functions. In this example, it requires solving a fairly simple linear program. In other settings, the problem may be more complex. For example, we may have a set of jobs and machines, and find that we have to solve a machine scheduling problem (but only for the jobs that we already know about at time t). In a logistics setting, we might have to schedule a fleet of vehicles serving a series of known demands (the machine scheduling problem and vehicle routing problem are quite similar). In the semiconductor industry, we might face the problem of deciding how to design batches of product (a similar problem in freight transportation involves deciding where to send a truck, and what packages to put on the truck).

Simulations that involve solving sequences of optimization models have been widely used in engineering practice, as well as in the research literature. Their biggest strength is their ability to use a cost function to make tradeoffs. But they ignore the impact of decisions now on the future. Are we using the right type of aircraft? If we send a job to a machine that requires a setup, how many jobs in the future will require the same setup? The issues are particularly important when we need to make decisions now in anticipation of demands in the future that have not arrived yet. If we have an idle machine, should we perform a setup because of jobs that might arrive in the future (given that we have the time now to do the setup)? Should we move a truck to a location that might need the truck (if we start the movement now, will we be better positioned to serve the demand)?

4 APPROXIMATE DYNAMIC PROGRAMMING

We now try to solve the optimization problem in (5) by identifying policies that make good decisions not just for now, but over time. In theory this can be done using dynamic programming, but in practice this has not proven to be practical. In this section, we introduce the emerging field of approximate dynamic programming, and describe a specific adaptation that works especially well for resource allocation problems. This adaptation overcomes some of the limitations of established algorithms known in the approximate dynamic programming literature (for good introductions, see Bertsekas and Tsitsiklis (1996) and Sutton and Barto (1998)).

4.1 The Optimality Equation and the Curses of Dimensionality

Since the 1950's it has been recognized that problems of the form (5) can be solved using Bellman's optimality equations, given by:

$$V_t(R_t) = \max_{x_t \in \mathcal{X}_t} C_t(R_t, x_t) + \gamma \sum_{r' \in \mathcal{R}} p(r'|R_t, x_t) V_{t+1}(r') \tag{10}$$

where $\mathcal{R}$ is the space of possible values of R_t, and $p(r'|R_t, x_t)$ is the probability that $R_{t+1} = r'$ given state R_t and action x_t. An equivalent form of this equation is

$$V_t(R_t) = \max_{x_t \in \mathcal{X}_t} C_t(R_t, x_t) + \gamma \mathbb{E}\{V_{t+1}(R_{t+1})|R_t\}, \tag{11}$$

where R_{t+1} is given by the transition function (4). The problem with computing either of these forms of the optimality equations has long been recognized as the "curse of dimensionality." If R_t is a vector, the number of possible values of R_t quickly becomes very large (problems with 10^{100} states are actually not that large in practice). As a result, computing (11) (or (10)) for each R_t is simply impossible.

In practice, the problem is much worse than this. We actually have three curses of dimensionality: the state space, the outcome space and the action space. Specifically, in addition to the large number of possible values of R_t, computing the expectation in (11) (the expectation is implicit in (10)) involves taking a sum over all possible values of the vector $\hat{R}_t$, which can be extremely large (in practical problems, $\hat{R}_t$ can have thousands or tens of thousands of dimensions). Finally, x_t is also a vector (again with thousands or tens of thousands of dimensions), eliminating the ability of using any algorithm that requires searching over all possible decisions.

4.2 An Algorithmic Framework

In practice, we have found that the most difficult problem we face when using (11) is computing (or even approximating) the expectation. We solve this by formulating the optimality equations around the post-decision state variable:

$$V^x_{t-1}(R^x_{t-1}) = \mathbb{E}\left\{\max_{x_t\in\mathcal{X}_t} C_t(R^x_{t-1},\hat{R}_t,x_t) + \gamma V^x_t(R^{M,x}(R^x_{t-1},\hat{R}_t,x_t))|R^x_{t-1}\right\}. \quad (12)$$

Readers may find it helpful to understand (12) by writing the optimality equation in terms of both pre- and post-decision state variables. We first break down the transition equation (4) into two steps. We may write the transition from pre-decision to post-decision state using:

$$R^x_t = R^{M,1}(R_t,x_t).$$

Then the transition from post-decision to pre-decision is given by:

$$R_{t+1} = R^{M,2}(R^x_t,\hat{R}_{t+1}).$$

Using this two-step transition, we may write the value functions for pre- and post-decision state variables using:

$$\begin{aligned} V_t(R_t) &= \max_{x_t\in\mathcal{X}_t}\left\{C_t(R^x_{t-1},\hat{R}_t,x_t) + \gamma V^x_t(R^{M,1}(R_t,x_t))\right\} \\ V^x_t(R^x_t) &= \mathbb{E}\left\{V^x_{t+1}(R^{M,2}(R_t,\hat{R}_{t+1}))|R^x_t\right\}. \end{aligned}$$

The concept of pre- and post-decision state variables are familiar to readers working with decision trees, where it is common to write decision nodes (where we choose an action) and outcome nodes (where we observe an exogenous event).

Our next step is to drop the expectation in (12) and write, for a specific sample realization $\hat{R}_t(\omega)$:

$$V^x_{t-1}(R^x_{t-1},\hat{R}_t(\omega)) = \max_{x_t\in\mathcal{X}_t(\omega)} C_t(R^x_{t-1},\hat{R}_t(\omega),x_t) + \gamma V^x_t(R^{M,x}(R^x_{t-1},\hat{R}_t,x_t)). \quad (13)$$

The key here is that given the post-decision state R^x_{t-1} and the information from time interval t, $\hat{R}_t(\omega)$, we can compute x_t by solving a deterministic function. Readers should verify that we cannot do this same trick using the pre-decision state variable.

The last step in our process is to replace the value function V^x_t (which we still do not know) with a suitable approximation which we denote, for the moment, $\bar{V}_t(R^x_t)$:

$$\tilde{V}{}^x_{t-1}(R^x_{t-1},\omega) = \max_{x_t\in\mathcal{X}_t(\omega)}\left\{C_t(R^x_{t-1},\hat{R}_t(\omega),x_t) + \gamma\bar{V}_t(R^x_t(x_t))\right\}.$$

$\tilde{V}{}^x_{t-1}(R^x_{t-1},\omega)$ is a placeholder. Assuming we can devise a reasonable approximation $\bar{V}_t(R^x_t)$, we now face the challenge of choosing what may be a high-dimensional decision vector x_t.

4.3 Approximation Strategies

The next step is the design of an approximation $\bar{V}_t(R^x_t(x_t))$. Our first consideration is computational. We want to design an approximation strategy that does not destroy any problem structure that our algorithm is exploiting. It is useful to start with the myopic problem:

$$x_t = \arg\max_{x_t\in\mathcal{X}_t} C_t(R^x_{t-1},\hat{R}_t,x_t).$$

Does this problem have any particular mathematical structure? Is it an assignment problem? A linear program? A shortest path problem? A machine scheduling problem? A vehicle routing problem? Think about how you would solve this problem. Are you using a linear programming package, or your favorite tabu search algorithm? Algorithms such as tabu search and genetic algorithms typically require little or no problem structure, whereas integer programming-based formulations often exploit problem structure.

The simplest approximation strategy is known as a "table lookup" function, which simply means that for any discrete state R_t, we find a function $\bar{V}_t(R_t)$ that produces an estimate of the value function given the state. This strategy requires that we estimate one parameter that gives the value of being in the state. Even small problems can have 10^{10} - 10^{100} states, so this is often impractical.

Another strategy is to use the structure of the problem to identify a regression model which is parameterized by a relatively small set of parameters θ_t. To do this, we start by identifying a set of *features* which appear to capture the important behaviors. For example, we might define:

$$\begin{aligned} \phi_f(R_t) &= \text{A function of the (resource) state variable, } f\in\mathcal{F}\text{, where:} \\ \mathcal{F} &= \text{The set of functions.} \end{aligned}$$

The set of features $(\phi_f(R_t))_{f\in\mathcal{F}}$ are often referred to as *basis functions*. Given a set of basis functions, we can

formulate a value function approximation using:

$$\bar{V}_t(R_t|\theta) = \sum_{f\in\mathcal{F}} \theta_f \phi_f(R_t). \quad (14)$$

In machine scheduling, one basis function might be the total number of jobs of a particular priority. Another might be the number of jobs needing a particular setup. In a transportation problem, we might define a basis function as the number of vehicles with a particular attribute:

$$\phi_a(R_t) = R_{ta}. \quad (15)$$

We call such functions "linear in the resource state" (to distinguish them from value functions which are linear in the parameters). We might also use:

$$\phi_a(R_t) = (R_{ta})^2.$$

We assume that the number of parameters, given by $|\mathcal{F}|$, is much smaller than the number of possible values of the state R_t. In both examples above, the number of basis functions is the same as the size of the attribute space $\mathcal{A}$, which will be much smaller than the number of states. If we are managing complex resources where $a \in \mathcal{A}$ may have a dozen (or several dozen) attributes, even $|\mathcal{A}|$ may be quite large. In this case, we may wish to aggregate the attribute space. Let $G(a)$ be a function that maps the original attribute vector a to a more compact attribute vector f (a feature f may be an index or a vector of elements). Our basis function might then be:

$$\phi_f(R_{tf}) = \sum_{a\in\mathcal{A}} R_{ta} 1_{\{G(a)=f\}}.$$

Of course, we may prefer to use a nonlinear relationship, such as:

$$\phi_f(R_t) = \sqrt{R_{tf}}, \quad (16)$$

where $R_{tf} = \sum_{a\in\mathcal{A}} R_{ta} 1_{\{G(a)=f\}}$. For discrete resource allocation problems, an effective approximation is to use piecewise linear functions:

$$\bar{V}_{ta}(R_{ta}) = \sum_{a\in\mathcal{A}} \left(\sum_{i=1}^{\lfloor R_{ta}\rfloor} \bar{v}_{tai} + (R_{ta} - \lfloor R_{ta}\rfloor)\bar{v}_{ta\lceil R_{ta}\rceil} \right), \quad (17)$$

where $\bar{v}_{ta} = (\bar{v}_{tai})_i$ is our vector of parameters to be estimated. Piecewise linear functions are useful when we are not comfortable using a linear function, but do not know the shape of the function.

The design of a value function approximation is the heart of any policy which wants to capture the impact of decisions now on the future. While there are some broad guiding principles and some specific tricks that can be used, for the most part the design of a value function approximation tends to be highly customized to the problem at hand. When designing the function, one question that should be resolved is whether the value function is needed to give you the value of being in a state, or the marginal value of another resource of a particular type. If you are solving the myopic problem using a linear (or integer) program, it is quite likely that you are more interested in the marginal value of another resource with a particular attribute. This is most apparent with the separable functions such as (15) or (16).

4.4 Estimation Strategies

Once we have defined the structure of the approximation, we next face the problem of actually estimating the parameters. There are a variety of techniques for doing this, but these can be organized along a few basic principles. All of them depend on using a Monte Carlo sample to create a random observation of the true value function, which is then used to update our approximation. We assume that we are at iteration n, following a sample path ω^n using value function approximations $\bar{V}_t^{n-1}(R_t)$ computed at iteration $n-1$. There are two classes of strategies for obtaining a Monte Carlo estimate of the true value function. One is to solve, for a given post-decision state $R_{t-1}^{x,n}$ and random sample $\hat{R}_t(\omega^n)$:

$$\hat{V}_t^n = \max_{x_t\in\mathcal{X}_t(\omega^n)} C_t(R_{t-1}^{x,n}, \hat{R}_t(\omega^n), x_t) + \gamma \bar{V}_t^{n-1}(R_t^x(x_t)). \quad (18)$$

Here, $\hat{V}_t^n$ is directly a function of $\bar{V}_t^{n-1}$. In the second method, we use the approximations $(\bar{V}_t^{n-1})_{t=1}^T$ to generate a sequence of decisions:

$$x_t^n = \arg\max_{x_t\in\mathcal{X}_t(\omega^n)} C_t(R_{t-1}^{x,n}, \hat{R}_t(\omega^n), x_t) + \gamma \bar{V}_t^{n-1}(R_t^x(x_t)), \quad (19)$$

for $t = 1, \ldots, T$. We compute $\hat{V}_t^n$ using:

$$\hat{V}_t^n = C_t(R_{t-1}^{x,n}, \hat{R}_t(\omega^n), x_t^n) + \gamma \hat{V}_{t+1}^n, \quad (20)$$

which is computed backward through time starting with $\hat{V}_{T+1}^n = 0$. Computing $\hat{V}_t^n$ using equation (18) requires a single forward pass through time, computing both decisions and sample values $\hat{V}_t^n$. In the second method, we compute the decisions x_t^n through a forward pass using (19), but then compute the $\hat{V}_t^n$ using a backward pass through equation (20).

The forward pass method for computing $\hat{V}_t^n$ (equation (18)) is easier to implement but generally provides slower convergence. The reason is that the approximations $\bar{V}_t^n$ are biased (because they are approximation and therefore are not the true values). If we use the forward/backward pass method ((19) to compute the decisions and (20) to compute $\hat{V}_t^n$), it is still the case that the observations $\hat{V}_t^n$ are biased, but at least they are unbiased samples of the value of following the policy determined by $\bar{V}^{n-1}$.

Once we have our Monte Carlo observations $\hat{V}_t^n$, we next have to update the value functions themselves. If we are using a simple table-lookup approximation (one value for each state) then we use

$$\bar{V}_{t-1}^n(R_{t-1}^{x,n}) = (1-\alpha^n)\bar{V}_{t-1}^{n-1}(R_{t-1}^{x,n}) + \alpha^n \hat{V}_t^n, \tag{21}$$

where α^n is a stepsize betwee 0 and 1. There is a broad range of stepsize formulas to choose from (see George and Powell (2005a) for a review), but a simple but effective class of strategies is of the form:

$$\alpha^n = \frac{a}{a+n^\beta - 1},$$

where a and β are parameters that have to be tuned ($a = 8$ and $\beta = .7$ are a good first choice for many problems).

If we are using a basis function representation, we can update the parameter vector θ using a simple formula. Let $\bar{\theta}^{n-1}$ be our current estimate of θ. The same principles that give us (21) for updating a table-lookup representation produce the following updating formula when using a basis function representation:

$$\begin{aligned}\bar{\theta}^n &= \bar{\theta}^{n-1} - \alpha^n(\bar{V}_t(R_t^n|\bar{\theta}^{n-1}) - \hat{V}_t^n)\nabla_\theta \bar{V}_t^{n-1}(R_t^n|\theta^n)\\ &= \bar{\theta}^{n-1} - \alpha^n(\bar{V}_t(R_t^n|\bar{\theta}^{n-1}) - \hat{V}_t^n)\phi(R_t^n),\end{aligned}$$

where $\phi(R_t^n)$ is a $|\mathcal{F}|$-element vector.

There is a broad range of resource allocation problems where we are more interested in the marginal value of increasing R_{ta} by one unit than we are in the total value $\hat{V}_t^n$. Let $\hat{V}_t^{n+}$ be computed just as we computed $\hat{V}_t^n$ (either with the forward pass, or forward/backward pass method), but instead of starting with $R_{t-1}^{x,n}$, we start with $R_{t-1}^{x,n} + 1$. For example, if we use the pure forward pass method, then

$$\begin{aligned}\hat{V}_{ta}^{n+} &= \max_{x_t \in \mathcal{X}_t(\omega^n)} C_t(R_{t-1}^{x,n} + e_{ta}, \hat{R}_t(\omega^n), x_t)\\ &\quad + \gamma \bar{V}_t^{n-1}(R_t^x(x_t)),\end{aligned}$$

where e_{ta} is a $|\mathcal{A}|$-dimensional vector of 0's with a 1 in the element corresponding to the element a. Next compute

$$\hat{v}_{ta}^n = \hat{V}_{ta}^{n+} - \hat{V}_t^n.$$

Finally, smooth these into estimates:

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

We note in passing that we use $\hat{v}_t^n$ (which uses sample information from time interval t) to estimate $\bar{v}_{t-1}^n$ which approximates an expectation of time t (and onward) using only the information available up through time $t-1$. This produces a linear-in-the-resource state value function approximation:

$$\bar{V}_{t-1}^n(R_{t-1}) = \sum_{a\in\mathcal{A}} \bar{v}_{t-1,a}^n R_{t-1,a}.$$

Computing $\hat{v}_{ta}^n$ takes advantage of the fact that finding $\hat{V}_{ta}^{n+}$ usually requires far less work than computing the initial $\hat{V}_t^n$. If the decision function is a linear program (this is the case with our airlift problem, or virtually any other linear resource allocation problem), we have to solve:

$$\max_{x_t\in\mathcal{X}_t} \sum_{a\in\mathcal{A}_t}\sum_{d\in\mathcal{D}} c_{tad}x_{tad} + \bar{V}^{n-1}(R_t^x(x_t)) \tag{22}$$

subject to:

$$\sum_{d\in\mathcal{D}^{ph}} x_{tad} = R_{ta}^A \quad a \in \mathcal{A}_t^{ph} \tag{23}$$

$$\sum_{a\in\mathcal{A}_t^{ph}} x_{tad} = R_{tb_d}^D \quad d \in \mathcal{D}_t^{D,ph} \tag{24}$$

$$x_{tad} \geq 0. \tag{25}$$

We first note that we can write

$$\begin{aligned}\bar{V}_t^{n-1}(R_t^x(x_t)) &= \sum_{a'\in\mathcal{A}} \bar{v}_{ta'}^{n-1} R_{ta'}^x(x_t)\\ &= \sum_{a'\in\mathcal{A}} \bar{v}_{ta'}^{n-1} \sum_{a\in\mathcal{A}}\sum_{d\in\mathcal{D}} \delta_{a'}(t,a,d)x_{tad}\\ &= \sum_{a\in\mathcal{A}}\sum_{d\in\mathcal{D}} x_{tad} \sum_{a'\in\mathcal{A}} \bar{v}_{ta'}^{n-1}\delta_{a'}(t,a,d)\\ &= \sum_{a\in\mathcal{A}}\sum_{d\in\mathcal{D}} x_{tad}\bar{v}_{t,a^M(t,a,d)}^{n-1}.\end{aligned}$$

Substituting the result into (22) gives:

$$\max_{x_t\in\mathcal{X}_t} \sum_{a\in\mathcal{A}_t}\sum_{d\in\mathcal{D}} \left(c_{tad} + \bar{v}_{ta^M(t,a,d)}^{n-1}\right) x_{tad}. \tag{26}$$

Solving (26) subject to (23)-(25) is a linear program. The dual variable for the flow conservation constraint (23) gives us an estimate of $\hat{v}_{ta}^n$ with no additional work. For more difficult problems, computing the incremental effect of an

additional resource is typically much easier than solving the problem from scratch.

Linear-in-the-resource state approximations can work well, but this is problem dependent. It is often the case that the value function is nonlinear in the resources. There are several strategies, but a particularly simple one takes advantage of knowledge of the problem. Assume that we can use our understanding of the problem to develop an initial approximation that we call $\bar{V}_t^n(R_t)$. For example, we might use

$$\bar{V}_{ta}^0(R_{ta}) = \theta_{ta}(R_{ta} - \bar{R}_{ta})^2,$$

where $\bar{R}_{ta}$ is a rough approximation of what we think might be the "right" amount of resources with attribute a (this can be computed at any level of aggregation). The SHAPE algorithm (Cheung and Powell 2000) updates the approximation using

$$\bar{V}_{ta}^n(R_{ta}) = \bar{V}_{ta}^{n-1}(R_{ta}) + \alpha^n(\hat{v}_{ta}^n - \nabla\bar{V}_{ta}^{n-1}(R_{ta}^n))R_{ta}.$$

The SHAPE algorithm is quite easy to use, but works best when the initial approximation $\bar{V}_{ta}^0(R_{ta})$ reflects a real understanding of the problem. All it does is tilt the original approximation with a linear correction term, but it has been found to provide very good results in certain problem settings.

4.5 How Well Does it Work?

In general, optimal solutions for stochastic, resource allocation problems of a practical size are simply unachievable. But there are several ways to obtain an indication of the quality of the solution. One strategy is to use these methods to solve a deterministic problem which can also be solved using a commercial solver. Godfrey and Powell (2002) consider a problem involving the management of fleets of identical vehicles which have to move loads of freight from one location to another (moving a vehicle from one location to another is comparable to performing a machine setup). The critical assumption is that each vehicle can serve one demand at a time, and that if a demand is not served at a particular point in time, it is lost.

This problem can be formulated as a linear program and solved using standard linear programming solvers. Table 1 compares the results using approximate dynamic programming to the optimal solution for problems with 20, 40 and 80 locations, over 15, 30 and 60 time period problems (for a complete description of the experiments, see Godfrey and Powell (2002)). The results indicate that the approximate solutions are near-optimal.

High-quality solutions for stochastic problems do not appear to be available. The best competition appears to be the most widely used engineering approximation, popularly

Table 1: Percentage of Optimal Solution (from LP Solver) Obtained using Piecewise Linear, Separable Approximations (from Godfrey and Powell 2002).

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

known as rolling horizon (or receding horizon) procedures. Here, we optimize at time t using what we know at time t plus a point forecast of future demands over a planning horizon. We implement only what is actionable at time t, and then rolling the clock forward one time period and repeat the procedure. For each problem, we can find the optimal solution assuming the entire future is known for a particular sample path. We refer to this solution as the posterior bound, since it is the best we could possibly do (optimizing with perfect information).

Figure 3 shows the results of the rolling-horizon procedure against that produced by the approximate dynamic programming solution, expressed as a percent of the posterior optimal solution. This figure is for a "single commodity" problem where the items being managed are all the same. Figure 4 does the same for a multicommodity problem, where heterogeneous resources are managed which can substitute for each other (at a cost) to serve the same demand. Both problems were run for a range of different assumptions about substitution costs (see Topaloglu and Powell (2005) for a more complete summary of the

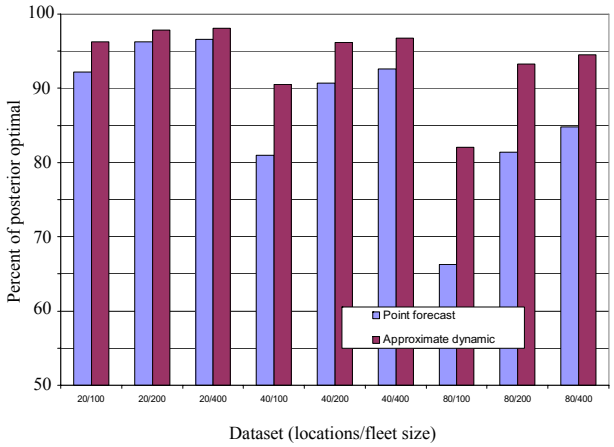


Figure 3: Percentage of Posterior Bound Produced by a Rolling Horizon Procedure using a Point Forecast of the Future Versus an Approximate Dynamic Programming Approximation on a Single Commodity Problem.

These results are hardly conclusive for all problems, but they are promising. At a minimum, they suggest that

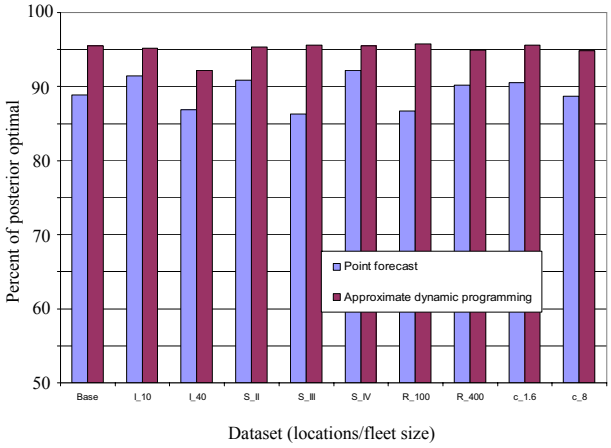


Figure 4: Percentage of Posterior Bound Produced by a Rolling Horizon Procedure Using a Point Forecast of the Future Versus an Approximate Dynamic Programming Approximation on a Multicommodity Problem.

relatively simple approximate dynamic programming strategies can offer significantly improved results over myopic policies or rolling horizon procedures.

4.6 Challenges

Approximate dynamic programming has been found to work extremely well for problems in fleet management that arise in trucking, rail and air. This said, it remains a relatively new technology which has to be tested in the context of specific applications. Technical challenges that should be anticipated include:

- Stepsizes - The choice of an appropriate rule for α^n can have a significant impact on the performance of the algorithm. If the stepsizes are too small, convergence can be extremely slow, and it is easy to conclude the algorithm does not work.
- Estimating $\hat{v}$ - One pass or two-passes? We often find that one-pass works well, but it can have slow convergence. If slow convergence is a problem, and you feel your stepsizes are large enough, try using a backward pass.
- Large attribute spaces - It is quite easy, even when managing relatively simple resources, for the attribute vector to grow as a project progresses, quickly producing an attribute space $\mathcal{A}$ that is far too large to enumerate. A common strategy is to use aggregation, which in this setting means estimating the value of a resource with an aggregated attribute vector. We have found that hierarchical aggregation (estimating the value of a resource at different levels of aggregation) works quite well (see George and Powell 2005b).
- Multiple resource layers - Our introduction here focused on *one layer* problems. For example, we captured the value of aircraft in the future, but not the demands. If we backlog demands, then we have the problem of jointly estimating the value of resources and demands in the future.

It is a good policy to focus initially on obtaining better results than you would with a myopic policy. If you can achieve this, look at your value function and see if you can find further improvements. The real issue is whether your value function approximation is capturing the downstream effects of decisions made now.

5 COMBINING COST-BASED AND RULE-BASED POLICIES

While obtaining optimal or near-optimal solutions is a worthy goal, we have to keep in mind that it is often the case that we are trying to simulate existing operations rather than make them better. In many problems, minimizing a cost function can be viewed as simulating what is typically rational, if imperfect, decision-making. However, it is also often the case that a cost model simply cannot capture all the complexities of real-world operations.

Consider the following examples:

- A truckload motor carrier wishes to assign its driver teams (two drivers in one truck) to longer loads because it takes advantage of the fact that the truck does not have to stop while the driver sleeps.
- The military may wish to avoid sending C-141's through airbases in a particular country because of the lack of maintenance facilities for this aircraft type.
- A printing company likes to send the print jobs for a particular customer to a specific printing plant because it is near the home location of the person who has to review the quality of the printing.
- A manufacturer prefers to have a specific machine handle certain jobs because the machine operator is more familiar with the nuances of the job which are not captured in the cost function.
- A railroad prefers to use high powered locomotives on certain types of trains that have to compete against fast moving trucks. It is possible to use lower-powered locomotives, but the slower times make the trains less competitive with a potential loss of revenue.

All of these represent operating policies that are likely to be expressed by a knowledgeable user. At the same time, they each introduce issues that are hard to quantify as a formal cost. Typically, it is fairly easy to express these

policies in a rule-based simulator, although even here it can be difficult to model statements such as "we prefer to" or "we like to avoid." These are not hard constraints; they are operating guidelines which the model should strive to achieve, recognizing that exceptions can occur.

Optimization modelers have long handled these issues by introducing artificial costs and bonuses to encourage or discourage certain behaviors. This process, however, is ad hoc and may require many repetitions to get the parameters right (increasing one cost to achieve one goal can interfere with the model's ability to achieve other goals).

These rules can often be expressed as low-dimensional patterns of behavior. Instead of complex "when in this state take this action" they are of the form "if we have this type of resource then take this type of decision." The "type of resource" can be modeled as a simplified attribute vector, that we denote here by $\bar{a}$. Also, the decision may also be more aggregate than an actual decision. Instead of "use this particular aircraft to move this particular load" it can be "use this type of aircraft to move this type of load." We also denote such aggregate decisions as $\bar{d}$. Our low dimensional patterns are given by:

$$\rho_{\bar{a}\bar{d}} = \text{The percent of time that we apply decisions of type } \bar{d} \text{ to resources with attributes of type } \bar{a}.$$

Let

$$\begin{aligned} G_a &= \text{Aggregation function that is applied to the attribute vector } a \in \mathcal{A}^A. \\ G_d &= \text{Aggregation function that is applied to the decision } d \in \mathcal{D}^A. \\ \mathcal{A} &= \{G_a(a) | a \in \mathcal{A}^A\}, \text{ the aggregated attribute space.} \\ \mathcal{D} &= \{G_d(d) | d \in \mathcal{D}^A\}, \text{ the aggregated decision space.} \end{aligned}$$

We now define a *pattern flow* as the vector of flows at the same level of aggregation as the pattern:

$$\begin{aligned} \bar{x}_{t\bar{a}\bar{d}} &= \sum_{a\in\mathcal{A}^A} \sum_{d\in\mathcal{D}_a^A} x_{tad} \cdot I_{\{G_a(a)=\bar{a}\}} \cdot I_{\{G_d(d)=\bar{d}\}} \\ \bar{R}_{t\bar{a}} &= \sum_{\bar{d}\in\mathcal{D}} \bar{x}_{t\bar{a}\bar{d}}, \end{aligned}$$

where $\bar{R}_{t\bar{a}}$ is the total aggregated flow with attribute $\bar{a}$ at time t. To compare our solution to the static pattern, we then define comparable static flows:

$$\begin{aligned} \bar{x}_{\bar{a}\bar{d}} &= \sum_t \bar{x}_{t\bar{a}\bar{d}} \\ \bar{R}_{\bar{a}} &= \sum_t \bar{R}_{t\bar{a}} \\ &= \sum_t \sum_{\bar{d}\in\mathcal{D}} \bar{x}_{t\bar{a}\bar{d}} \\ \bar{\rho}_{\bar{a}\bar{d}}(x) &= \bar{x}_{\bar{a}\bar{d}}/\bar{R}_{\bar{a}} \\ \bar{\rho}(x) &= (\bar{\rho}_{\bar{a}\bar{d}}(\bar{x}))_{\bar{a}\in\mathcal{A},\bar{d}\in\mathcal{D}}. \end{aligned}$$

We try to match the pattern flow by adding a penalty term to our objective function, given by

$$H(\bar{\rho}(x), \rho) = \sum_{\bar{a}\in\mathcal{A}} \sum_{\bar{d}\in\mathcal{D}} \bar{R}_{\bar{a}} (\bar{\rho}_{\bar{a}\bar{d}}(\bar{x}) - \rho_{\bar{a}\bar{d}})^2.$$

$H(\bar{\rho}(\bar{x}), \rho)$ is known as the *pattern metric*. We add it to the cost function to obtain the modified objective for our decision function:

$$x_t(\theta) = \arg\max_{x_t\in\mathcal{X}_t} \sum_{a\in\mathcal{A}^A} \sum_{d\in\mathcal{D}_a^A} c_{tad} x_{tad} - \theta H(\bar{\rho}(x), \rho),$$

where θ is a scaling factor. Normally, adding a separable penalty term does not complicate the algorithmic solution. When we are managing discrete assets, it is useful to replace the quadratic term with a piecewise linear term, with breakpoints at integer values for flows.

The more significant problem is that our patterns are static (we do not have patterns that tell us what to do at time t), but decisions are dynamic. We can handle this effectively using a Gauss-Seidel strategy where we compute the pattern metric at time stage t by combining decisions $x_{t'}^n$ for $t' < t$ with decisions from the previous iteration $x_{t'}^{n-1}$ for $t' > t$. These are held fixed at time t while we are optimizing only over x_t^n. For more details, we refer the reader to Powell, Wu and Whisman (2004) and in particular to Marar and Powell (2005).

Matching patterns requires that the simulator be run iteratively. This is also required if we use approximate dynamic programming, so both can be performed at the same time. Our experience is that pattern matching works after only three or four iterations.

An illustration of this idea is given in Figure 5. In this experiment, we were given a pattern (expressed as expert knowledge) that specified the percent of time that aircraft moving through a particular airbase was of a particular type. Without any guidance, the model would use this aircraft approximately 18 percent of the time. We then varied this pattern from 0 to 100, with different values for the penalty weight θ. As θ approached 1000, the pattern matching

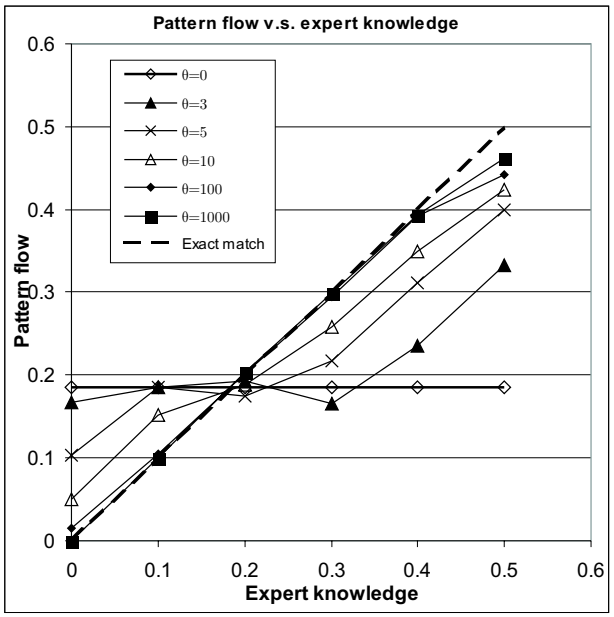


Figure 5: Percent of Time a Particular Activity Happened against the Pattern Prescribing the Fraction of Time the Activity Should Happen, for Different Values of θ (from Powell, Wu and Whisman 2004).

logic would almost perfectly match the desired pattern. For smaller values, the fraction of time that we used this aircraft at this airbase would increase as we increased the pattern, but would not necessarily match the desired pattern. This experiment indicates how the pattern matching logic can encourage desirable behavior from the model.

6 CONCLUDING REMARKS

The optimizing-simulator is a modeling and algorithmic strategy that combines the ability of math programming to handle high-dimensional resource allocation problems with the flexibility of simulation to handle uncertainty and complex system dynamics. It requires formulating a decision function which may be myopic (rule-based or cost-based), or may use two forms of adaptive learning.

The first form of adaptive learning uses approximate value functions so that decisions now can use at least an approximate estimate of the impact of decisions on the future. For fairly simple resource allocation problems, it has been shown that suitably chosen value function approximations can produce near-optimal solutions.

The second recognizes that cost-based objective functions are often imperfect, resulting in decisions that are quickly criticized by a knowledgeable expert. These criticisms can often be expressed as low-dimensional patterns. We encourage our model to match these patterns by penalizing deviations from the patterns. Since these patterns are almost always static, we have to run the model iteratively to adjust our time-dependent behavior to match these average patterns.

By retaining a cost-based objective function, we can easily handle fairly complex tradeoffs that are difficult to capture in a rule-based decision function. Such logic is critical for more complex resource allocation problems (which are common in transportation and logistics) which require the power of algorithms drawn from the field of math programming. However, our experience implementing these models with companies has shown us that cost-based models often do not produce desired behaviors. Low-dimensional patterns help retain some of the flexibility to simply tell the model what to do.

In conclusion, when deciding whether to simulate or optimize, the answer appears to be to combine both technologies.

ACKNOWLEDGMENTS

This research was supported in part by grant AFOSR-FA9550-05-1-0121 from the Air Force Office of Scientific Research and NSF grant CMS-0324380. The paper also benefited from a number of helpful comments by Douglas Morrice for which we are very appreciative.

REFERENCES

Baker, S., Morton, D., Rosenthal, R. and Williams, L. 2002. Optimizing military airlift, *Operations Research* 50(4): 582–602.

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

Cheung, R. K.-M. and Powell, W. B. 2000. SHAPE: A Stochastic Hybrid Approximation Procedure for Two-Stage Stochastic Programs. *Operations Research*, 48(1): 73–79. Available online at <http://www.castlelab.princeton.edu/Papers/>

George, A. and Powell, W. B. 2005a. Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming, Technical report, Department of Operations Research and Financial Engineering, Princeton University. Available online at <http://www.castlelab.princeton.edu/Papers/>

George, A. and Powell, W. B. 2005b. Value Function Approximation using Hierarchical Aggregation for Multiattribute Resource Management, Department of Operations Research and Financial Engineering, Princeton University. Available online at <http://www.castlelab.princeton.edu/Papers/>

Godfrey, G. and 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. Available online at <http://www.castlelab.princeton.edu/Papers/>

Killingsworth, P. and Melody, L. J. 1997. Should C-17's be deployed as theater assets?: An application of the conop air mobility model Technical report rand/db-171-af/osd, Rand Corporation.

Marar, A. and Powell, W. B. 2005. Using static flow patterns in time-staged resource allocation problems. Technical report, Princeton University, Department of Operations Research and Financial Engineering. Available online at <http://www.castlelab.princeton.edu/Papers/>

Powell, W. B., Wu, T. T. and Whisman, A. 2004. Using low dimensional patterns in optimizing simulators: An illustration for the airlift mobility problem, *Mathematical and Computer Modeling* 29: 657–2004.

Rosenthal, R., Morton, D., Baker, S., Lim, T., Fuller, D., Goggins, D., Toy, A., Turker, Y., Horton, D. and Briand, D. 1997. Application and extension of the Thruput II optimization model for airlift mobility, *Military Operations Research* 3(2): 55–74.

Schank, J., Mattock, M., Sumner, G., Greenberg, I. and Rothenberg, J. 1991. A review of strategic mobility models and analysis, Technical report, RAND Corporation.

Sutton, R. and Barto, A. 1998. *Reinforcement Learning*, The MIT Press, Cambridge, Massachusetts.

Topaloglu, H. and Powell, W. B. (2005), 'Dynamic programming approximations for stochastic, time-staged integer multicommodity flow problems', *Informs Journal on Computing* (to appear). Available online at <http://www.castlelab.princeton.edu/Papers/>

AUTHOR BIOGRAPHY

WARREN B. POWELL is a professor in the department of Operations Research and Financial Engineering at Princeton University. He is director of CASTLE Laboratory and has implemented optimizing-simulator models in both military and civilian settings, including a number of the largest freight transportation companies in the U.S. The coauthor of over 100 refereed publications, he has focused on solving complex stochastic resource allocation problems, and has recently focused on merging the fields of simulation, math programming and approximate dynamic programming.