*Proceedings of the 2007 Winter Simulation Conference*
*S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, eds.*

# Approximate Dynamic Programming for High-Dimensional Problems

Warren B. Powell

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

## ABSTRACT

There is a wide range of simulation problems that involve making decisions during the simulation, where we would like to make the best decisions possible, taking into account not only what we know when we make the decision, but also the impact of the decision on the future. Such problems can be formulated as dynamic programs, stochastic programs and optimal control problems, but these techniques rarely produce computationally tractable algorithms. We demonstrate how the framework of approximate dynamic programming can produce near-optimal (in some cases) or at least high quality solutions using techniques that are very familiar to the simulation community. The price of this challenge is that the simulation has to be run iteratively, using statistical learning techniques to produce the desired intelligence. The benefit is a reduced dependence on more traditional rule-based logic.

## 1 INTRODUCTION

There is a vast range of problems that can be described under the banner of "resource allocation." Many of these problems involve decisions that have to be made over time under various forms of uncertainty, and often involve fairly complex physical processes. These are the problems that tend to fall in the domain of Monte Carlo simulation.

A challenge that arises in many resource allocation problems is the dimensionality of the decisions being made. Consider the problem of allocating $I$ types of resources (blood, money, people, fuel, equipment) to $J$ types of demands. If we let $x_{ij}$ be the number of resources of type $i$ assigned to demands of type $j$, we find ourselves with a problem that is most naturally formulated as a linear program. This is fairly easy if we are only solving a problem at one point in time; the difficulty arises when we want to solve the problem over time. A common modeling strategy is to assume that events (e.g., demands) in the future are known, and solve a single linear program over a planning horizon. The "simulation" of activities in the future is handled by the optimization algorithm.

These problems have created a tension over the years between the simulation community, which promotes its ability to not only handle uncertainty but also a variety of complex operational considerations, and optimization, which focuses on its ability to produce high quality solutions. A common strategy is to simulate an optimization problem by stepping forward in time, solving sequences of optimization problems based on what is known at a point in time. At time $t$, we can solve an optimization problem using only what is known at time $t$, or using a deterministic forecast of future events ("rolling horizon procedures"). While either strategy can produce good results in specific settings, both introduce serious weaknesses in many applications.

It is possible to combine the power of simulation and optimization using the framework of approximate dynamic programming (ADP). The result is a method that is accurately described as an "optimizing simulator." ADP is easily adapted to existing simulators, since it steps forward in time as with any simulation model. The only difference is that as the simulation runs, we collect information statistically that is then used to improve the quality of decisions. This information captures the impact of decisions now on the future.

## 2 SAMPLE APPLICATIONS

Every technique is characterized by applications that best take advantage of its features. Approximate dynamic programming is well suited to problems which involve some combination of: complex dynamics, uncertainty, high-dimensional decision vectors and a need to make decisions that take into account the impact on the future.

Applications where ADP offers tremendous promise include:

- Transportation problems
  - *Management of freight cars:* Railroads have to decide how many freight cars to move from one location to another to meet future random demands. The model has to account for variability in equipment types and random travel times.
  - *Locomotive scheduling:* A train typically requires several locomotives with specific characteristics to move a train. Plans have to be made up to a week into the future. In addition to a number of complex operational constraints, planning has to consider last-minute additions and cancellations, locomotive failures and train delays.
  - *Driver assignment:* Trucking companies have to assign drivers to move loads of freight, responding to changes in customer requests as well as a variety of operating rules (getting drivers home, observing limits on driving hours, putting drivers on the right types of loads).
- Storage problems
  - *Natural gas storage:* How much gas should be purchased and stored in large coal mines during low demand periods to be sold when demands (and prices) are high? Gas can be purchased from multiple sources, stored in multiple locations, and sold using forward contracts at different times of the year.
  - *High-value spare parts:* Airlines, electric power utilities, manufacturing operations and medical suppliers are a small number of examples of operations which have to maintain inventories of high-value spare parts. These are stored in small numbers (often 0 or 1) in various locations around the country to respond to infrequent, sporadic demands.
  - *Cash balance optimization:* Mutual funds have to maintain a cash balance to respond to redemption requests. The amount of cash has to reflect not only random demands but also market conditions.
- Financial applications
  - *Portfolio planning:* How much money should be invested in each of a set of random investment opportunities? Depending on the type of investment, we may have to consider transaction times as well as transaction costs.
  - *Pricing complex options:* There are a variety of instruments that allow the sale or purchase of assets in the future. Determining the value of these options requires finding the best policy for exercising the option.
  - *R & D portfolio management:* The government has to determine which research activities to support to accomplish a specific goal (e.g., energy independence).
- Energy applications
  - *Acquiring new energy assets:* Governments and companies need to determine how much capacity they should have to create energy from different sources. This requires understanding the complex dynamics of wind, solar and hydrothermal power to make decisions to serve future (and highly uncertain) demands in the presence of changing technologies.
  - *Control of power generating facilities:* Regional transmission organizations (which manage the electric power grid) have to determine which generating plants (coal, gas, nuclear) to turn on and off each day to respond to daily and seasonal cycles for power.
  - *Energy inventories:* Companies have to determine how much oil/coal/natural gas/biomass to produce, where to store it and how to get it there, all in an environment of random market prices and demands.
- Military applications
  - *Movement of cargo aircraft:* The military airlift command has to manage a fleet of cargo aircraft to move freight and people. The models have to handle complex constraints on the movement and storage of aircraft at airbases, refueling and maintenance, as well as weather and equipment problems.
  - *Mid-air refueling:* The air force has to plan the movement of tankers to handle the demands of aircraft ("receivers") performing various military operations. A model has to track the fuel level of tankers and receivers, as well as a number of operational constraints on how the receivers interact with the tankers.
  - *Management of UAV's:* Unmanned aerial vehicles are often used to collect information about regions, ranging from the status of targets to movements of people.
- Demand management
  - *Hospital admissions:* How many patients can be admitted for elective surgery given available beds, operating rooms and medical staff?

  - *Booking hotel space:* Hotels have to book meetings taking into consideration available hotel rooms, banquet rooms and meeting rooms.
  - *Load acceptance:* Trucking companies and railroads have to determine when to make commitments to customers given available resources in the future.
- Manufacturing applications
  - *Routing in queueing networks:* In a flexible manufacturing facility, it is necessary to determine which machine a part should be routed to after it has finished a step, given the queues at different machines within the facility.
  - *Control of reconfigurable servers:* A machine might be set up to paint parts a particular color, drill a particular type of hole, or fill a particular type of bottle. From time to time, machines can be converted to perform a different function. The problem is determining when to switch each machine from one function to another.
- Medical applications
  - *Blood management:* How much blood of a particular type and age should be used now versus held for the future?
  - *Scheduling medical personnel:* Performing drug trials requires scheduling medical personnel, equipment and facilities to serve patients.
  - *Allocating antivirals for flu outbreaks:* It is necessary to determine how many antivirals to allocate to people with specific characteristics during each week of the flu season, given the current status of the disease within the population.

All of these problems involve different forms of uncertainty, and some introduce complex operational details that are most easily handled using simulation. They also all introduce an opportunity to make decisions that balance rewards now against rewards in the future. Our goal is to do a better job of making decisions that balance the "here and now" against the future. There are multiple reasons for wanting to do this. In addition to the obvious goal of wanting better solutions, there are applications where this behavior is needed to more realistically match actual system behavior. After all, when people make decisions, they typically consider events in the future.

## 3 A GENERAL MODEL

We begin by providing a generic model of a resource allocation problem. Our model is hardly the most general, but it provides sufficient richness to handle the problems described above, and helps to illustrate the elements of approximate dynamic programming and the relationship to both simulation and optimization.

We assume that we are managing "resources" to serve "demands." These are modeled using

$$\begin{aligned}
a &= \text{Vector of attributes describing a resource, where } a \in \mathscr{A},\\
b &= \text{Vector of attributes describing a demand, where } b \in \mathscr{B},\\
R_{ta} &= \text{The number of resources with attribute } a \in \mathscr{A} \text{ in the system at time } t,\\
R_t &= (R_{ta})_{a\in\mathscr{A}},\\
D_{tb} &= \text{The number of demands of type } b \in \mathscr{B} \text{ in the system at time } t,\\
D_t &= (D_{tb})_{b\in\mathscr{B}}.
\end{aligned}$$

Often, we are modeling the problem in the presence of random parameters that govern the evolution of the system. These parameters might be prices, weather, the cost of a technology (the cost of solar panels) or the performance of a technology (the efficiency of a solar panel). We represent these parameters generically using

$$\rho_t = \text{A generic vector of parameters that affects the behavior of costs and the transition function.}$$

A basic state variable might be given by

$$S_t = (R_t, D_t, \rho_t).$$

Above, we assume that $t$ represents a point in time at which a decision is made. While decisions are made in discrete time, we model information as arriving in continuous time. Information arrives in the form of exogenous changes to our state variable. For our problem, we might have three types of exogenous information processes:

$$\begin{aligned}
\hat{R}_{ta} &= \text{Exogenous changes to } R_{ta} \text{ from information that arrives during time interval } t \text{ (between } t-1 \text{ and } t),\\
\hat{D}_{tb} &= \text{Exogenous changes to } D_{ta} \text{ from information that arrives during time interval } t \text{ (between } t-1 \text{ and } t).\\
\hat{\rho}_t &= \text{Exogenous changes to a vector of parameters (costs, parameters governing the transition).}
\end{aligned}$$

The random variable $\hat{R}_{ta}$ might be used to capture equipment failures or delays. Similarly, the random variable $\hat{D}_{tb}$ might represent a new customer demand or a change in the attributes of an existing demand. $\hat{\rho}_t$ would capture changes in costs or performance (e.g., due to research). Exogenous information

can be modeled generically using

$$W_t = (\hat{R}_t, \hat{D}_t, \hat{\rho}_t).$$

Note that at time $t$, $W_t$ is known, while $W_{t+1}$ is unknown. This choice of indexing helps to resolve what is and is not known at a point in time.

We model decisions using

$\mathscr{D}^D$ = Decision to satisfy a demand with attribute $b$ (each decision $d \in \mathscr{D}^D$ corresponds to a demand attribute $b_d \in \mathscr{B}$).

$\mathscr{D}^M$ = Decision to modify a resource (each decision $d \in \mathscr{D}^M$ has the effect of modifying the attributes of the resource). $\mathscr{D}^M$ includes the decision to "do nothing."

$\mathscr{D}$ = $\mathscr{D}^D \cup \mathscr{D}^M$.

$x_{tad}$ = The number of resources that initially have attribute $a$ that we act on with decision $d$.

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

Decisions have to satisfy basic constraints on the availability of resources and the number of demands to be served. This is done using

$$\sum_{d\in\mathscr{D}} x_{tad} = R_{ta}, \tag{1}$$

$$\sum_{a\in\mathscr{A}} x_{tad} \leq D_{tb_d} \quad d \in \mathscr{D}^D, \tag{2}$$

$$x_{tad} \geq 0. \tag{3}$$

In a particular application, other constraints might arise. For this reason, we let $\mathscr{X}_t$ be the feasible region for the vector $x_t$. For now, this consists of equations (1) - (3), but additional constraints may be included in specific applications.

At the heart of our problem is the need to make a decision. For the moment, we assume this is accomplished by a decision function, given by

$X_t^\pi(S_t)$ = A function that returns a decision vector $x_t \in \mathscr{X}_t$, where $\pi \in \Pi$ is an element of the set of functions (policies) $\Pi$.

Once we have made a decision, we model the evolution of the system using a classical transition function which we represent generically using

$$S_{t+1} = S^M(S_t, x_t, W_{t+1}).$$

Our notation reflects by the tradition in some communities to refer to this equation as the *system model*, *plant model* or simply *model*. For many problems in resource allocation, it is useful to introduce a specific model that governs the evolution of the attributes of a specific resource (which might be a person, facility or piece of equipment). We represent this function using

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

There are many problems where this is deterministic. For example, the random information $W_{t+1}$ might include only information about the demands, while the evolution of the resource, once a decision is made, is deterministic. For algebraic purposes, we define

$$\delta_{a'}(a,d) = \begin{cases} 1 & \text{if } a' = a^M(a_t, d_t, W_{t+1}), \\ 0 & \text{otherwise.} \end{cases}$$

Using this notation, we can write the transition function for the resource vector $R_t$ using

$$R_{t+1,a'} = \sum_{a\in\mathscr{A}} \sum_{d\in\mathscr{D}} \delta_{a'}(a,d) x_{tad} + \hat{R}_{t+1,a'}. \tag{4}$$

The demand transition function is given by

$$D_{t+1,b_d} = D_{tb_d} - \sum_{a\in\mathscr{A}} x_{tad} + \hat{D}_{t+1,b_d} \quad d \in \mathscr{D}^D. \tag{5}$$

Remember that for each $d \in \mathscr{D}^D$, there is a demand of type $b_d \in \mathscr{B}$. This function assumes that unserved demands are held for the future. For many problems, unserved demands are lost, in which case $D_{t+1,b} = \hat{D}_{t+1,b}$. The evolution of our technology vector evolves according to

$$\rho_{t+1} = \rho_t + \hat{\rho}_{t+1}. \tag{6}$$

Equations (4) - (6) constitute our transition function $S^M(S_t, x_t, W_{t+1})$.

Finally, we have to specify our objective function. For our resource allocation problems, we define a contribution (cost if we are minimizing) given by

$c_{tad}$ = Contribution earned (negative if it is a cost) from using decision $d$ acting on resources with attribute $a$.

Assuming a linear contribution function, the total contribution would be given by

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

Our problem is to find a decision function that solves

$$\max_{\pi\in\Pi} \mathbb{E}\left\{ \sum_{t=0}^{T} C_t\left(S_t, X_t^\pi(S_t)\right) \right\}. \tag{7}$$

## 4 THE DECISION FUNCTION

For complex problems, the transition function $S^M(\cdot)$ can be quite difficult to formulate, but we are going to assume that this is given. Our challenge is designing a decision function, which in some communities is referred to as a decision rule or, more commonly, a policy.

There are two broad categories of decision functions: rule-based and cost-based. Optimization models exclusively use cost-based decisions (by definition), where the "cost" can be a contribution, reward or utility to be maximized, or a cost or penalty to be minimized. The field of discrete event simulation primarily uses rule-based decisions.

Rule-based policies come in several forms:

- Look-up table - If we are in discrete state $s$, the table gives us a discrete action to take.
- Parameterized rules - If the inventory is less than $q$, order up to $Q$. Assign a job leaving one machine to the highest priority machine that has available buffer space (the parameter is the buffer space).
- Regression policies - If we have $R$ gallons of water in the reservoir, release $x_t = \theta_1 R_t + \theta_2 R_t^2 + \theta_3 \ln R_t$, where $\theta$ is a vector of parameters to be determined.

There are numerous strategies for determining these policies, but they tend to be very problem-specific. In special cases, these policies can produce optimal or near-optimal solutions to (7), but often the interest is simply modeling an existing strategy or operation. An example is a simulator used by the airlift mobility command to model the movements of cargo aircraft. At any point in time, there is a list of available aircraft (sorted by time of availability) and a list of "requirements" (loads of freight or people) to be moved (also sorted by time order). Their current simulator uses a rule that starts with the first requirement to be moved, then looks at the first available aircraft (regardless of location) to see if the assignment is feasible (e.g., there is capacity to move the aircraft through intermediate airbases). There is no attempt to assess a cost for any action.

Cost-based policies also come in different forms:

- Myopic policies - Here we make decisions consider only on the contribution we earn right now, as in

$$X^\pi(S_t) = \arg\max_{x_t \in \mathscr{X}_t} C(S_t, x_t).$$

- Rolling horizon policies - Here we choose decisions $x_t, x_{t+1}, \ldots, x_{t+P}$ over a planning horizon $P$, using a deterministic forecast of events in the future. Normally we implement only $x_t$, after which we sample new information ($W_{t+1}$) and repeat the process for $t+1$.
- Dynamic programming policies - This approach (which is the focus of this paper) makes a decision now using

$$X_t^\pi(R_t) = \arg\max_{x_t \in \mathscr{X}_t} \left(C(S_t, x_t) + \gamma \mathbb{E} V(S_{t+1})\right) \quad (8)$$

where $S_{t+1} = S^M(S_t, x_t, W_{t+1})$ and $\gamma$ is a discount factor. The challenge here is finding the function $V(S_{t+1})$.

Myopic policies ignore the impact of decisions now on the future, which will produce poor solutions for many of the problems we are interested in. Rolling horizon procedures work well for many problems, but they depend on a deterministic forecast of the future, which can perform very poorly. Also, rolling horizon procedures can be computationally demanding, since at each point in time you have to solve a problem over a horizon $t, \ldots, t+P$. Aside from producing a potentially poor solution (since it ignores uncertainty), the resulting problem may be quite large and therefore difficult to solve.

In the remainder of this paper, we focus on using the framework of dynamic programming to produce good decisions. The foundation of dynamic programming is Bellman's equation, typically written in the form

$$V_t(S_t) = \max_{x_t \in \mathscr{X}_t} \left( C(S_t, x_t) + \gamma \sum_{s'} p(s'|S_t, x_t) V_{t+1}(s') \right) \quad (9)$$

$$= \max_{x_t \in \mathscr{X}_t} \left(C(S_t, x_t) + \gamma \mathbb{E}\left\{V_{t+1}(s')|S_t\right\}\right), \quad (10)$$

where $p(s'|S_t, x_t)$ is the probability that we transition to state $S_{t+1} = s'$ given that we are in state $S_t$ and take action $x_t$. Equation (9) is the form most commonly seen in textbooks (e.g., Puterman (1994)) while (10) is the mathematically equivalent *expectation form*. As a solution strategy, Bellman's equation is typically dismissed due to the "curse of dimensionality" which produces exponentially large state spaces when $S_t$ is a vector.

In our applications, there are actually three curses of dimensionality: the state space (the state variable may be a vector), the outcome space (the number of outcomes of $W_t$, which complicates computing the expectation in (10)), and the action space (the number of potential actions $x_t$ in $\mathscr{X}_t$). Approximate dynamic programming provides a framework for using dynamic programming in a simple and elegant way within a simulation model.

## 5 APPROXIMATE DYNAMIC PROGRAMMING

Approximate dynamic programming has been evolving since the 1950's from within the artificial intelligence community (Samuel 1959), and the early work of Bellman himself

(Bellman and Dreyfus 1959). The field evolved primarily within the artificial intelligence community and the control theory/neural network community under names such as reinforcement learning and neuro-dynamic programming. The field really emerged in the 1990's with the appearance of two major books (Bertsekas and Tsitsiklis (1996) and Sutton and Barto (1998)) and edited volumes (Miller, Sutton, and Werbos (1990) and White and Sofge (1992)). The merger of dynamic programming and stochastic approximation theory was established in 1994 by Tsitsiklis (1994) and Jaakkola, Jordan, and Singh (1994). The merger of approximate dynamic programming and math programming took place through a series of papers (Godfrey and Powell (2001a), Papadaki and Powell (2003), Powell and Van Roy (2004), Powell (2005)) and a recent book (Powell 2007).

We take several steps to overcome the three curses of dimensionality. The first and most critical is the use of the *post-decision state variable* which measures the state of the system immediately after a decision has been made (but before any time has passed which would bring new information). There are different ways to define a post-decision state variable. The one that we use assumes that we can break out the pure effect of a decision from the pure effect of new information. We do this using

$$\begin{aligned} S_t^x &= \text{The state at time } t \text{ immediately after a decision has been made,} \\ &= S^{M,x}(S_t, x_t), \\ S_{t+1} &= \text{The pre-decision state at time } t, \\ &= S^{M,W}(S_t^x, W_{t+1}). \end{aligned}$$

The post-decision state takes on different forms which are highly problem dependent, but we can illustrate using a simple inventory (storage) problem. Let $R_t$ be the quantity being stored (water, natural gas, spare equipment). We assume this is the quantity just before we make a decision $x_t$ to order more (we can model sales if we allow $x_t$ to be negative). We call $R_t$ the pre-decision state, and the post-decision state is given by

$$R_t^x = R_t + x_t.$$

Now let $\hat{D}_{t+1}$ the demand we have to satisfy in the next time period. The next pre-decision state is

$$R_{t+1} = \max(0, R_t^x - \hat{D}_{t+1}).$$

Another way of representing a post-decision state is to assume that we have access to a point estimate of the information that will arrive in the next time period. Let

$$\bar{W}_{t,t+1} = \text{A point estimate of the information that will arrive between } t \text{ and } t+1.$$

We can now define our pre- and post-decision states using

$$\begin{aligned} S_t^x &= S^M(S_t, x_t, \bar{W}_{t,t+1}) \\ S_{t+1} &= S^M(S_t, x_t, W_{t+1}). \end{aligned}$$

Thus, $S_t^x$ can be viewed as a type of forecast (more precisely, a point estimate) of $S_{t+1}$ that we make at time $t$ using the decision $x_t$ and a forecast of the future information.

Using the post-decision state, we can break Bellman's equations into two steps:

$$V_t(S_t) = \max_{x_t \in \mathscr{X}_t} \left( C(S_t, x_t) + \gamma V_t^x(S_t^x) \right) \tag{11}$$

$$V_t^x(S_t^x) = \mathbb{E} V_{t+1}(S_{t+1}) \tag{12}$$

where $S_t^x = S^{M,x}(S_t, x_t)$ and $S_{t+1} = S^M(S_t, x_t, W_{t+1})$ and the expectation is over the outcomes of the random variable $W_{t+1}$. If we substitute (12) into (11), we obtain the classical form of Bellman's equation. It is important to note that (11) is a pure deterministic optimization problem, while (12) is purely an expectation.

The problem is that for the vast majority of applications, we will not know $V_t^x(S_t^x)$. As a result, we have to replace it with an approximation which we represent by $\bar{V}_t(S_t^x)$. We never need to compute (even approximately) $V_t(S_t)$. Using $\bar{V}_t(S_t^x)$, our decision function looks like

$$X_t^\pi(S_t) = \arg\max_{x_t \in \mathscr{X}_t} \left( C(S_t, x_t) + \gamma \bar{V}_t(S_t^x) \right). \tag{13}$$

The critical step is designing $\bar{V}_t(S_t^x)$, which depends very much on the nature of the problem and the available algorithms. For example, there are many resource allocation problems where $x_t$ represents a schedule for discrete equipment (machines, locomotives, cargo aircraft), or an allocation of continuous resources (blood, money, vaccines). In the first case, we may have to use an integer programming package or possibly a metaheuristic such as tabu search or a genetic algorithm. In the second, we probably face a linear or nonlinear programming problem.

A good way to approach the problem is to first ask how you would solve the problem if there were no value function (can you enumerate actions? can you use a linear or integer programming code? is it a nonlinear programming problem?). Your answer will determine the type of structure you want to retain in your value function. For example, if you need to use a linear, nonlinear or integer programming package, you are going to need some sort of continuous value function approximation. If you are going to use a search algorithm such as tabu search or a genetic algorithm, then you can use a look-up table (where you have a value $\bar{V}_t(S)$ for each discrete state $S$).

Once we find a value function approximation, we have to determine how to estimate it. The simplest is a lookup-

table where there is a value for each state $S$. Further assume that we are in post-decision state $S_{t-1}^{x,n}$ at time $t-1$, when we are in iteration $n$ of our algorithm. Let $W_t(\omega^n)$ be a sample realization of the random variable $W_t$, representing a sample of the information that arrives between $t-1$ and $t$. Our next pre-decision state would be

$$S_t^n = S^{M,W}(S_{t-1}^{x,n}, W_t(\omega^n)).$$

We then solve

$$\hat{v}_t^n = \max_{x_t \in \mathscr{X}_t^n} \left(C_t(S_t^n, x_t) + \gamma \bar{V}_t^{n-1}\left(S^{M,x}(S_t^n, x_t)\right)\right) \quad (14)$$

where $\bar{V}_t^{n-1}(S^{M,x}(S_t^n, x_t))$ is the value function approximation from the previous iteration. We then update the value function using

$$\bar{V}_{t-1}^n(S_{t-1}^{x,n}) = (1-\alpha_{n-1})\bar{V}_{t-1}^{n-1}(S_{t-1}^{x,n}) + \alpha_{n-1}\hat{v}_t^n,$$

where $\alpha_{n-1}$ is a stepsize that is typically between 0 and 1. There are many recipes for stepsizes (see George and Powell (2006)), the simplest popular ones are a constant (such as 0.1) or a declining sequence such as $a/(a+n)$ where $a$ is some constant such as 5 or 10. Note that we use $\hat{v}_t^n$, which is an estimate of the value of being in pre-decision state $S_t^n$, to update the value function around the previous post-decision state $S_{t-1}^{x,n}$.

An overall summary of the algorithm is given in Figure 1 for a finite horizon problem. The same algorithm can be adapted for infinite horizon problems by dropping the time-index on the value function approximation.

## 6 VALUE FUNCTION APPROXIMATIONS

In general, we will not be able to use look-up table representations for the value function simply because there are too many states. Instead, we use various approximation strategies. We address two dimensions of the approximation problem that arise in the context of resource allocation: the quantity problem (estimating the value of $R > 1$ resources) and the quality problem (what is the difference between a resource with attribute vector $a'$ or $a''$). We start with the quantity problem.

### 6.1 The quantity problem

Often we have to determine whether to have three doctors, 400 freight cars, 10 million dollars, or 50 units of blood. The approximations below help in the design of value functions where we have to determine the quantity of resources. All of these can be used within an optimization package for applications where $x_t$ is a (possibly high-dimensional) vector.

**Step 0.** Initialization:

- **Step 0a.** Initialize $\bar{V}_t^0, \quad t \in \mathscr{T}$.
- **Step 0b.** Set $n = 1$.
- **Step 0c.** Initialize $S_0^1$.

**Step 1.** Choose a sample path $\omega^n$.

**Step 2.** Do for $t = 0, 1, 2, \ldots, T$:

- **Step 2a.** Solve:

$$\hat{v}_t^n = \max_{x_t \in \mathscr{X}_t^n} \left(C_t(S_t^n, x_t) + \gamma \bar{V}_t^{n-1}(S^{M,x}(S_t^n, x_t))\right)$$

  and let $x_t^n$ be the best value of $x_t$.
- **Step 2b.** If $t > 0$, update the value function:

$$\bar{V}_{t-1}^n \leftarrow U^V(\bar{V}_{t-1}^{n-1}, S_{t-1}^{x,n}, \hat{v}_t^n)$$

- **Step 2c.** Update the states:

$$S_t^{x,n} = S^{M,x}(S_t^n, x_t^n)$$
$$S_{t+1}^n = S^{M,W}\left(S_t^{x,n}, W_{t+1}(\omega^n)\right)$$

**Step 3.** Increment $n$. If $n \le N$ go to Step 1.

**Step 4.** Return the value functions $(\bar{V}_t^N)_{t=1}^T$.

Figure 1: Generic approximate dynamic programming algorithm using the post-decision state

#### Linear approximations

The simplest approximation is one that is linear in the resource state, given by

$$\bar{V}_t(S_t^x) = \sum_{a \in \mathscr{A}} \bar{v}_{ta} R_{ta}^x.$$

We emphasize that the state variable (more precisely, the post-decision state) $S_t^x$, may consist of not only the resource state $R_t^x$ but also other forms of information (weather, prices, technology). We assume here that the value function approximation is purely a function of $R_t^x$.

To estimate the slopes $\bar{v}_{ta}$, we use the derivative of the decision function rather than the value of being in a particular state. That is, rather than use the value of being in a state (as we did in equation (14)), we are going to use the derivative of the objective function

$$\tilde{V}_t(R_t^n) = \max_{x_t \in \mathscr{X}_t^n} \left(C(S_t^n, x_t) + \gamma \bar{V}_t(S_t^x)\right). \quad (15)$$

where $S_t^x = S^{M,x}(S_t^n, x_t)$. Let

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

be the derivative of $\tilde{V}_t(R_t^n)$ with respect to the element $R_{ta}$. If the optimization problem in (15) is a linear program, we can obtain an estimate of the derivative using the dual variable of the resource constraint in (1). We then smooth our estimate of the value using

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

One value of working with derivatives is that instead of getting one estimate of the value of being in a state, we get a vector of derivatives. This is exceptionally powerful.

**Separable, piecewise linear approximations**

In many applications, the marginal value of a resource depends on the quantity of resources, since there are often declining marginal returns. A simple and flexible way of capturing this behavior is through the use of separable, piecewise linear approximations. We would write the value function approximation using

$$\bar{V}_t(S_t^x) = \sum_{a \in \mathscr{A}} \bar{V}_{ta}(R_{ta}^x).$$

Often, we use a simpler attribute vector in the value function. Thus, a locomotive might be characterized by location, locomotive type, home shop and fuel level, while in the value function we may only consider location and locomotive type.

Updating piecewise linear value function approximations is very similar to updating linear value functions. Instead of updating a single slope for resources with attribute $a$, you update the slope around the point $R_{t-1,a}^{x,n}$ corresponding to the previous post-decision state variable. The problem is that smoothing $\hat{v}^n$ with the current slope at $\bar{V}_{t-1,a}^{n-1}(R_{t-1,a}^{x,n})$ may produce a function that is no longer concave. The steps are illustrated in Figure 2. We start with a piecewise linear, concave function $\bar{V}_{t-1,a}^{n-1}(R_{t-1,a}^{x,n})$. The slope $\hat{v}^n$ is shown as a dashed line, from which we update the value function by smoothing $\hat{v}^n$ with the current slope at $R = R^n$. This produces a piecewise linear value function that is no longer concave. We then have to introduce a step to retain concavity. There are several ways of doing this, including the CAVE algorithm (Godfrey and Powell 2001b), the leveling algorithm (Topaloglu and Powell 2003) and the SPAR algorithm (Powell, Ruszczyński, and Topaloglu 2004). All are quite simple to implement.

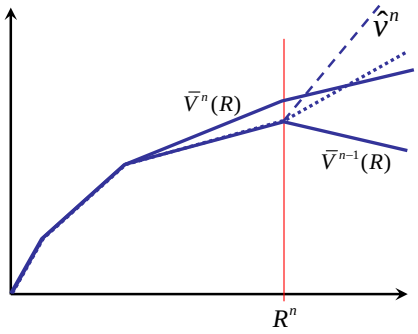


Figure 2: Original value function $\bar{V}^{n-1}(R)$ (solid line), estimate of slope at $R^n$ which violates concavity, and updated value function $\bar{V}^n(R)$ after concavity has been maintained

**Indexed separable, piecewise linear approximations**

This is a hybrid of piecewise linear, separable and lookup-table. Let $\phi_f(S_t)$ be some statistic computed from the state variable (this could be a measure of weather, an indicator of the state of solar panel technology, the market price of oil). We assume there is a small set of these statistics (called features), given by $\mathscr{F}$. Let $\phi(S_t) = \left(\phi_f(S_t)\right)_{f \in \mathscr{F}}$ be the set of features, where we assume they have been discretized into a set $\Phi_t$ which we hope is not too large. Our value function approximation would then be

$$\bar{V}_t(S_t^x) = \sum_{a \in \mathscr{A}} \bar{V}_{ta}\left(R_{ta}^x | \phi(S_t)\right).$$

Now, instead of one function for each $a \in \mathscr{A}$, we have $|\Phi_t|$ functions. The steps for updating the piecewise linear functions are the same as the scalar case, but now we may have dozens, hundreds or even thousands of functions for each attribute. This introduces additional statistical challenges. A successful illustration of this strategy is given in Nascimento and Powell (2007).

There are two important special cases of this strategy. In the first, $\phi(S_t)$ depends purely on exogenous factors such as weather, technology or market prices. In the second, $\phi(S_t) = \phi(R_t)$ depends directly on the (pre-decision) resource vector.

**Polynomial approximations**

Another strategy is to use polynomial approximations. For example, we might specify a linear regression of the form

$$\bar{V}(R) = \sum_{a \in \mathscr{A}} \left(\theta_{ta}^0 + \theta_{ta}^1 R_{ta} + \theta_{ta}^2 (R_{ta})^2\right).$$

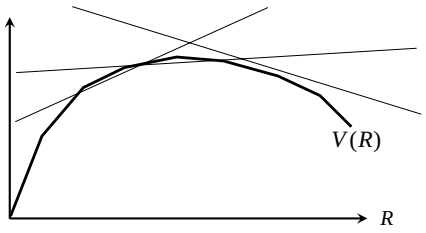


Figure 3: Illustration of Benders cuts to approximate a concave function

We can write this more generally as

$$\bar{V}(S) = \sum_{f \in \mathscr{F}} \theta_f \phi_f(S).$$

Here, the functions $\phi_f(S)$ are referred to as *basis functions* in the approximation literature, or *features* in the reinforcement learning literature. They are functions which extract information which is felt to be relevant in explaining the impact of the state variable on the value function.

The estimation of basis functions has received considerable attention in the approximate dynamic programming community. Important references are Bertsekas and Tsitsiklis (1996), Tsitsiklis and Van Roy (1996), Tsitsiklis and Van Roy (1997), Van Roy (2001) and Powell (2007).

**Benders cuts**

This is an approximation drawn from the stochastic programming community. Here, the value function is represented using a series of cuts of the form

$$\bar{V}_t(R_t^x) \leq \alpha_m^n + \beta_m^n R_t^{x,m}, \quad m = 1, \ldots, n.$$

An illustration of the use of a series of cutting planes to approximation a concave function is given in Figure 3.

There are several strategies for estimating the parameters $\alpha_m^n$ and $\beta_m^n$, the most popular being stochastic decomposition Higle and Sen (1991). Powell (2007) provides a brief summary of the use of Benders cuts in the context of approximate dynamic programming. Unlike the other methods, Benders cuts produces provably optimal solutions under certain conditions, but experimental work (Powell, Ruszczyński, and Topaloglu (2004), Topaloglu and Powell (2006)) suggests that the rate of convergence can be quite slow.

### 6.2 The quality problem

Estimating the value of $R$ resources, for $R > 1$, primarily arises in resource allocation problems where the attribute space is small. When managing complex resources (people, complex equipment), we encounter what can be called the quality problem: what is the value of resources of type $a'$ versus $a''$? If $a = (a_1, a_2, \ldots, a_I)$, where $I$ is more than three or four attributes, the attribute space gets large very quickly. Even if we just use a linear approximation, estimating $\bar{v}_{ta}$ can become statistically difficult. We may need to estimate hundreds of thousands of parameters (sometimes far more). Even when $\hat{v}_{ta}$ represent derivatives, we are not going to be able to compute an estimate of the derivative for every attribute at every iteration.

An effective way of handling this problem is to estimate values at different levels of aggregation (see Powell (2007) and George, Powell, and Kulkarni (2005)). Assume that we are given a set of functions $G^g$, $g \in \mathscr{G}$ which map the attribute space $\mathscr{A}$ into more compact representations, which we write

$$G^g : \mathscr{A} \rightarrow \mathscr{A}^{(g)}.$$

Let $\bar{v}_a^{(g,n)}$ is an estimate of the attribute $a \in \mathscr{A}$ at the $g^{\text{th}}$ level of aggregation (that is, for the attribute $G(a) \in \mathscr{A}^{(g)}$). We can estimate $\bar{v}_a^{(g,n)}$ simply from aggregate observations using

$$\bar{v}_a^{(g,n)} = (1 - \alpha_{n-1})\bar{v}_a^{(g,n-1)} + \alpha_{n-1}\hat{v}_a^n \quad \text{for each } g \in \mathscr{G}.$$

We then create a single estimate of the value of a resource with attribute $a$ using

$$\bar{v}_a^n = \sum_{g \in \mathscr{G}} w_a^{(g,n)} \bar{v}_a^{(g,n)}.$$

A simple way of estimating the weights is to make them inversely proportional to the sum of the variance and bias squared

$$w_a^{(g,n)} \propto (\bar{\sigma}_a^2)^{(g,n)} + \left(\bar{\mu}_a^{(g,n)}\right)^2,$$

where $(\bar{\sigma}_a^2)^{(g,n)}$ is an estimate of $\bar{v}_a^{(g,n)}$ and $\bar{\mu}_a^{(g,n)}$ is an estimate of the bias, given by

$$\bar{\mu}_a^{(g,n)} = \bar{v}_a^{(g,n)} - \bar{v}_a^{(0,n)}.$$

We compute $(\bar{\sigma}_a^2)^{(g,n)}$ by first finding

$$\begin{aligned}
\bar{v}_a^{(g,n)} &= (1 - \eta_{n-1})\bar{v}_a^{(g,n-1)} + \eta_{n-1}(\bar{v}_a^{(g,n-1)} - \hat{v}_a^n)^2, \\
\bar{\beta}_a^{(g,n)} &= (1 - \eta_{n-1})\bar{\beta}_a^{(g,n-1)} + \eta_{n-1}(\hat{v}^n - \bar{v}_a^{(g,n-1)}), \\
(s_a^2)^{(g,n)} &= \frac{\bar{v}_a^{(g,n)} - (\bar{\beta}_a^{(g,n)})^2}{1 + \lambda^{n-1}}.
\end{aligned}$$

where

$$\lambda_a^{(g,n)} = \begin{cases} (\alpha_{a,n-1}^{(g)})^2 & n = 1 \\ (1-\alpha_{a,n-1}^{(g)})^2 \lambda_a^{(g,n-1)} + (\alpha_{a,n-1}^{(g)})^2 & n > 1. \end{cases}$$

Here, we have written the stepsize as $\alpha_{a,n-1}^{(g)}$ to emphasize that it depends on the attribute and the level of aggregation. $\eta_{n-1}$ is a simple stepsize such as 0.05. Finally, we compute the variance of $\bar{v}_a^{(g,n)}$ using

$$\begin{aligned} (\bar{\sigma}_a^2)^{(g,n)} &= Var[\bar{v}_a^{(g,n)}] \\ &= \lambda_a^{(g,n)} (s_a^2)^{(g,n)} \end{aligned} \tag{16}$$

These equations are easy to implement and scale to large attribute spaces. Note that we do not have a problem if we have no observations at a level of aggregation, since in this case we simply set the weight to zero.

## 7 FROM SIMULATION TO OPTIMIZATION

So you already have a simulation model of your favorite application. Perhaps it is a queueing network, a hospital application, or a transportation problem. Are you a candidate for approximate dynamic programming?

The first question you have to answer: do you have an objective function that provides a single, quantitative measure that determines the quality of the decisions you are making? If not, you are not a candidate for optimization. If so: Do you use a rule-based policy for making decisions, or do you maximize a contribution or minimize a cost? If you use rule-based decision making, you need to make the conversion to a contribution (cost)-based one.

As a simple illustration, the military uses a rule-based simulation to model the flows of cargo aircraft. Given a load of freight, the model tries to assign the first available aircraft, without regard to location. An alternative is to look at the first five available aircraft (or all the aircraft that are available within a specific horizon), determine a cost for assigning each aircraft to the load, and then choose the one with the lowest cost. This is a small step, but a significant one.

Once you have a way of making decisions by maximizing a contribution, we then have to ask: do you need to capture the impact of decisions now on the future? Just as important, what sort of information do you need to make better decisions? In one transportation project involving the assignment of drivers to loads, we needed approximate dynamic programming to tell us what *type* of driver should be assigned to a load. In a project managing freight cars, we needed value functions to tell us *how many* cars should be moved to a location. Some problems are dominated by the immediate impact of a decision, while others produce meaningless results if you do not think about the future.

After you have decided the type of intelligent behavior you are looking for, the next task is to choose a value function approximation that captures this. This involves three steps: identifying the elements of the state variable that are important, creating a value function approximation that works with the technology that you are using to find a decision (a linear programming solver?, a tabu search heuristic?), and finally, designing a method to estimate the value function.

Designing a value function approximation is part art (what is important?) and part science (it has to work with your solver, you need to update it, and it has to have good statistical properties). It is not unusual to obtain poor results when you first start testing an ADP algorithm. As you progress through your testing, make sure you address the following questions:

- Is $\hat{v}_{ta}^n$ being properly calculated? If the system is taking you to a state that seems to produce poor results, does $\hat{v}_{ta}^n$ reflect this?
- Be careful with stepsizes. A common mistake is to use a stepsize $\alpha_n = 1/n$ since this has been proven to produce convergent results. In fact, it can work extremely poorly. Start with $\alpha_n = 0.10$ or 0.05. As you build confidence (the results seem to improve, but not as well as you hoped), switch to $\alpha_n = a/(a+n)$ for $a = 5$ or 10. Later, switch to an adaptive stepsize formula such as the "optimal stepsize algorithm" (George and Powell 2006) (presented as the "bias-adjusted Kalman filter" stepsize in Powell (2007)).
- Be aware of the "exploration vs. exploitation" problem. There are problems where you have a poor value of a state because you have not visited the state. You do not visit the state because your estimate of the value of the state is low. See chapter 10 in Powell (2007) for a more complete discussion.

## ACKNOWLEDGMENTS

This research was supported in part by grant AFOSR-FA9550-05-1-0121 from the Air Force Office of Scientific Research.

## REFERENCES

Bellman, R., and S. Dreyfus. 1959. Functional approximations and dynamic programming. *Mathematical Tables and Other Aids to Computation* 13:247–251.

Bertsekas, D., and J. Tsitsiklis. 1996. *Neuro-dynamic programming*. Belmont, MA: Athena Scientific.

George, A., and W. B. Powell. 2006. Adaptive stepsizes for recursive estimation with applications in approximate

dynamic programming. *Machine Learning* 65 (1): 167–198.

George, A., W. B. Powell, and S. Kulkarni. 2005. Value function approximation using hierarchical aggregation for multi-attribute resource management. Technical report, Princeton University, Department of Operations Research and Financial Engineering.

Godfrey, G. A., and W. B. Powell. 2001a. 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.

Godfrey, G. A., and W. B. Powell. 2001b. 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.

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

Jaakkola, T., M. I. Jordan, and S. P. Singh. 1994. Convergence of stochastic iterative dynamic programming algorithms. In *Advances in Neural Information Processing Systems*, ed. J. D. Cowan, G. Tesauro, and J. Alspector, Volume 6, 703–710. San Francisco: Morgan Kaufmann Publishers.

Miller, W. T. I., R. S. Sutton, and P. J. Werbos. (Eds.) 1990. *Neural networks for control*. Cambridge, MA: MIT Press.

Nascimento, J., and W. B. Powell. 2007. Dynamic programming models and algorithms for the mutual fund cash balance problem. Technical report, Princeton University.

Papadaki, K., and W. B. Powell. 2003. An adaptive dynamic programming algorithm for a stochastic multiproduct batch dispatch problem. *Naval Research Logistics* 50 (7): 742–769.

Powell, W. B. 2005. The optimizing-simulator: Merging simulation and optimization using approximate dynamic programming. In *Proceedings of the Winter Simulation Conference*. New York: OMNIPress.

Powell, W. B. 2007. *Approximate dynamic programming: Solving the curses of dimensionality*. New York: John Wiley and Sons.

Powell, W. B., A. Ruszczyński, and H. Topaloglu. 2004. Learning algorithms for separable approximations of stochastic optimization problems. *Mathematics of Operations Research* 29 (4): 814–836.

Powell, W. B., and B. Van Roy. 2004. Approximate dynamic programming for high dimensional resource allocation problems. In *Handbook of Learning and Approximate Dynamic Programming*, ed. J. Si, A. G. Barto, W. B. Powell, and D. W. II. New York: IEEE Press.

Puterman, M. L. 1994. *Markov decision processes*. New York: John Wiley & Sons.

Samuel, A. L. 1959. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development* 3:211–229.

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

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

Topaloglu, H., and W. B. Powell. 2006. Dynamic programming approximations for stochastic, time-staged integer multicommodity flow problems. *Informs Journal on Computing* 18 (1): 31–42.

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

Tsitsiklis, J. N. 1994. Asynchronous stochastic approximation and Q-learning. *Machine Learning* 16:185–202.

Tsitsiklis, J. N., and B. Van Roy. 1996. Feature-based methods for large scale dynamic programming. *Machine Learning* 22:59–94.

Van Roy, B. 2001. Neuro-dynamic programming: Overview and recent trends. In *Handbook of Markov Decision Processes: Methods and Applications*, ed. E. Feinberg and A. Shwartz. Boston: Kluwer.

White, D. A., and D. A. Sofge. 1992. *Handbook of intelligent control*. New York, NY: Von Nostrand Reinhold.

## 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 specialized in solving complex stochastic resource allocation problems, and has recently focused on merging the fields of simulation, math programming and approximate dynamic programming.