Approximate Dynamic Programming for High Dimensional Resource Allocation Problems

Warren B. Powell
Abraham George
Belgacem Bouzaiene-Ayari
Hugo P. Simao
Department of Operations Research
and Financial Engineering
Princeton University
Princeton, NJ 08544
E-mail: powell@princeton.edu

***Abstract*— There are a wide array of discrete resource allocation problems (buffers in manufacturing, complex equipment in electric power, aircraft and locomotives in transportation) which need to be solved over time, under uncertainty. These can be formulated as dynamic programs, but typically exhibit high dimensional state, action and outcome variables (the three curses of dimensionality). For example, we have worked on problems where the dimensionality of these variables is in the ten thousand to one million range. We describe an approximation methodology for this problem class, and summarize the problem classes where the approach seems to be working well, and research challenges that we continue to face.**

I. Introduction

There is a vast range of discrete resource allocation problems that arise in engineering and operational problems. How large should the buffers be in a manufacturing assembly line? How many transformers should a utility purchase, what type, and where should they be stored when they are built? How should unmanned aerial vehicles (UAV's) be allocated to best collect information in a military setting? How should cargo aircraft be scheduled to respond to dynamic demands, weather problems and equipment failures?

These problems are often most naturally formulated as discrete dynamic programs, but these can be very difficult to solve. The dimensionality of the state variables can easily range from a few dozen, to several thousand to a million or more, producing state spaces on the order of $10^{10} - 10^{100}$ or more. The dimensionality of the control variable can be much larger. The dimensionality of the random variables representing new information are of a similar size.

This paper describes a class of strategies in the field of approximate dynamic programming that are showing promise for solving these large-scale problems.

The paper is organized as follows. We begin in Section II by presenting a simple but general model for the management of discrete resources. Section III describes an algorithmic strategy using the principles of approximate dynamic programming. Section IV reports on some numerical work from previous research that suggests that the methods are able to solve realistic problems to near-optimality. Section V closes with some research challenges.

II. A basic model

We can describe a very general class of discrete resource allocation problems using a fairly simple modeling framework. Throughout the paper, $t = 0$ represents "here and now." The interval from $t = 0$ to $t = 1$ is indexed as time interval 1. Decisions are assumed to be made in discrete time, but we view information as arriving continuously over time. Thus, W_t may be the information that arrives during time interval t, but x_t is the decision made at time t. Any variable indexed by t is assumed to have access to information that has arrived up through time t (which is to say that it is $\mathcal{F}_t$-measurable, using common probability conventions).

The resources are modeled using:

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

The attribute vector might include location, the type of equipment, other attributes such as age, repair status, and skills (for people). In many applications, we may know about a resource that we cannot use. For example, we may have placed an order for a piece of equipment that might arrive days or months into the future. A piece of transportation equipment may be in the process of moving between two locations. In these cases, the attribute vector captures the attributes when the resource can be used, and includes as an attribute the time at which it can be used.

Decisions are represented using:

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

At a minimum, the feasible region $\mathcal{X}_t$ captures the flow conservation constraint:

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

We assume the set $\mathcal{D}$ always includes a decision that allows us to "do nothing" to a resource (this is the reason why equation (1) is written as an equality). In some problems, a decision d represents a decision to purchase a particular type of resource, including its features and characteristics, when it will arrive and where it will be located when it does arrive. In other problems, we have resources that we can act on (changing location, repairing, converting, turning off or on). It is useful to represent the effect of a decision using the indicator variable:

$$\delta_{a'}(t,a,d) = \begin{cases} 1 & \text{If decision } d, \text{ acting on a resource with attribute } a, \text{ produces a resource with attributes } a'. \\ 0 & \text{Otherwise} \end{cases}$$

This notation allows us to write the resource dynamics using:

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

$$R_{t+1,a'} = R^x_{t,a'} + \hat{R}_{t+1,a'} \tag{3}$$

We refer to R^x_t as 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.

For simplicity, we assume that costs are linear, allowing us to define:

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

Without specifying how a decision is made, we define a family of decision functions (policies):

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

Our challenge is to find the best policy $(X^\pi_t(R_t))_{\pi\in\Pi}$ that solves:

$$\sup_{\pi\in\Pi} \mathbb{E}\left\{\sum_{t=0}^{T} \gamma^t C_t(X^\pi_t(R_t))\right\} \tag{4}$$

where γ is a discount factor. We can write more general decision functions that depend on more than just the resource state variable R_t, but this problem is enough to illustrate our ideas. Solving (4) exactly is typically intractable, so we propose a class of dynamic programming approximations in the next section.

III. An approximate dynamic programming algorithm

In this section, we first outline the basic approximation strategy, which requires using a nonstandard version of the optimality equations. Next we describe different approximation strategies for the value function, and finally describe how these approximations can be updated at each iteration. For an introduction to the basic ideas behind approximate dynamic programming, see references (1) and (2).

A. Optimality recursion using post-decision state variable

In theory, we can solve our optimization problem using the optimality equations:

$$V_t(R_t) = \max_{x_t\in\mathcal{X}_t} C_t(x_t) + \gamma\mathbb{E}\left\{V_{t+1}(R_{t+1})|R_t\right\} \tag{5}$$

where R_{t+1} is a function of x_t (see (2) - (3)) and the new information $\hat{R}_{t+1}$. It is important to keep in mind that R_t may have hundreds, thousands or tens of thousands of dimensions. The allocation vector x_t is usually larger. The information vector $\hat{R}_{t+1}$ is of similar dimensions. To provide an indication of how large the state space is, if $N = \sum_{a\in\mathcal{A}} R_{ta}$ is the number of resources we are managing, if $\mathcal{R}$ is the space of outcomes of R_t, the number of possible values that the vector R_t can take on is given by:

$$|\mathcal{R}| = \begin{pmatrix} N+|\mathcal{A}|-1 \\ |\mathcal{A}|-1 \end{pmatrix} \tag{6}$$

Not surprisingly, this state space gets very large, very quickly even for moderate N. In transportation applications, we might be managing thousands of people or pieces of equipment, with attribute spaces that can reach into the thousands or hundreds of thousands.

We start by recognizing that the myopic problem:

$$\max_{x_t\in\mathcal{X}_t} C_t(x_t) \tag{7}$$

is typically a linear or integer program that can be solved using commercially available solvers. Since we are allocating discrete resources, we have to solve an integer program, but it is often the case that ignoring integrality still produces optimal solutions that are integer. This occurs because many discrete resource allocation problems exhibit network or near-network structure, producing integer optimal solutions. Losing this property can significantly complicate the solution of these problems. Not surprisingly, we would like to approximate our problem in a way that retains the structure of the myopic problem.

Our approximation strategy requires several steps. First, the optimality equation written in (5) is formulated around R_t,

which we have referred to as the pre-decision state variable. While this is most common, we are going to use the optimality equation formulated around the post-decision state variable R_t^x. This is given by:

$$V_{t-1}^x(R_{t-1}^x) = \mathbb{E}\left\{\max_{x_t \in \mathcal{X}_t} C_t(x_t) + \gamma V_t^x(R_t^x(x_t))|R_{t-1}^x\right\} \quad (8)$$

We next drop the expectation and solve the problem for a single realization of the information $\hat{R}_t$ that would be known when we make the decision x_t:

$$V_{t-1}^x(R_{t-1}^x, \omega) = \max_{x_t \in \mathcal{X}_t(\omega)} C_t(x_t, \omega) + \gamma V_t^x(R_t^x(x_t)) \quad (9)$$

Finally, we replace the value function with a conveniently chosen approximation (to be discussed below):

$$\tilde{V}{}_{t-1}^{x}(R_{t-1}^x, \omega) = \max_{x_t \in \mathcal{X}_t(\omega)} C_t(x_t, \omega) + \gamma \bar{V}_t(R_t^x(x_t)) \quad (10)$$

where $\bar{V}_t(R_t)$ is our approximation, and $\tilde{V}{}_{t-1}^{x}(R_{t-1}^x, \omega)$ is a placeholder. At this point we have eliminated the problematic expectation (one of our curses of dimensionality). The key now is to choose a suitable continuous approximation for $\bar{V}_t(R_t)$ that allows us to solve (10). The idea is to then use information derived from solving this problem to update the approximation $\bar{V}_{t-1}$.

B. Approximation strategies

A general approximation strategy that has been widely studied in approximate dynamic programming is to start with a set of *basis functions* which are calculations using the state variable that produce results that are felt to explain the behavior of the system (see references (3), (4)). We start by defining a set of basis functions:

$$\begin{aligned} \phi_b(R_t) &= \text{A function of the (resource) state variable,} \\ &\quad b \in \mathcal{B}, \text{ where:} \\ \mathcal{B} &= \text{The set of basis functions} \end{aligned}$$

We might then approximate our value function using:

$$\bar{V}_t(R_t|\theta) = \sum_{b \in \mathcal{B}} \theta_b \phi_b(R_t) \quad (11)$$

where $\theta = (\theta_b)_{b \in \mathcal{B}}$ is a set of parameters to be determined (for time-dependent problems, θ may also be time-dependent).

A basis function may be as simple as the number of resources with a particular attribute vector:

$$\phi_a(R_t) = R_{ta} \quad (12)$$

For more complex resources with perhaps a dozen or more attributes, it is natural to define aggregations on the attribute vector. Let $G(a)$ represent an aggregation of a, and let $G : \mathcal{A} \rightarrow \mathcal{A}^g$, where $\mathcal{A}^g$ is the aggregated attribute space. We can define a set of basis functions where $\mathcal{B} = \mathcal{A}^g$, and define the basis function using:

$$\phi_b(R_t) = \sum_{a \in \mathcal{A}} R_{ta} 1_{\{G(a)=b\}}$$

In many resource allocation problems, the value of additional resources declines with the number of resources. For example, in manufacturing, bigger buffers reduce blockages, but there are declining marginal returns. Having more transportation equipment may increase revenues, but again, there are typically declining marginal returns. With this insight, we may conclude that a good basis function also exhibits declining returns, and we may use functions such as:

$$\phi_a(R_t) = \sqrt{R_{ta}} \quad (13)$$

If we use the value function in (11) with basis functions such as those given by (12) or (13), then we would say that our basis function is linear in the parameters. But we might decide that our function has declining marginal returns, but we are not sure that a square root function is the best. Instead we might propose:

$$\phi_a(R_t) = (R_{ta})^{-\beta_a}$$

where the vector $\beta = (\beta_a)_{a \in \mathcal{A}}$ is another vector of parameters to be estimated. Now we have a value function approximation which is nonlinear in the parameters.

For discrete resource allocation problems, a particularly powerful strategy is to use piecewise linear functions:

$$\bar{V}_{ta}(R_t) = \sum_{a \in \mathcal{A}} \left(\sum_{i=1}^{\lfloor R_{ta} \rfloor} \bar{v}_{tai} + (R_{ta} - \lfloor R_{ta} \rfloor)\bar{v}_{ta\lceil R_{ta} \rceil} \right) \quad (14)$$

where $\bar{v}_{ta} = (\bar{v}_{tai})_i$ is our vector of parameters to be estimated. The separable, piecewise linear approximation in (14) does not fit the structure of a basis function, so we use slightly different notation. The parameter $\bar{v}_{tai}$ is an approximation of the slope of the value function when $R_{ta} = i$. For the special case of a linear approximation (where we mean linear in the resource state vector), we propose the notation:

$$\bar{V}_t = \sum_{a \in \mathcal{A}} \bar{v}_a R_{ta}$$

In both (15) and (14), the parameters to be estimated both represent approximations of the slope of the value function. We exploit this observation in the next section when we describe an updating strategy.

C. Estimation strategies

If we use a linear or piecewise linear approximation structure, solving problems of the form:

$$\max_{x_t \in \mathcal{X}_t(\omega)} C_t(x_t, \omega) + \gamma \bar{V}_t(R_t^x(x_t)) \quad (15)$$

subject to, for all $a \in \mathcal{A}$:

$$\sum_{d \in \mathcal{D}_a} x_{tad} = R_{t-1,a} + \hat{R}_{ta}(\omega) \quad (16)$$

$$x_{tad} \geq 0 \quad (17)$$

means solving a linear program. We may have additional constraints, but we must have flow conservation (16) and

nonnegativity (17). When we solve (15)-(17) using a linear programming package, we obtain dual variables for equation (16). Let $\hat{v}_{ta}$ be the dual variable for the resource constraint R_{ta}. Remember that this is a random variable since it depends on the outcome ω, producing a realization of the information vector $\hat{R}_{ta}(\omega)$.

We can use $\hat{v}_{ta}$ to help estimate the expected marginal value of an additional resource. Assume that our current estimate (after the $n-1^{st}$ iteration) of the slope of the value of $R_{t-1,a}$ is $\bar{v}^{n-1}_{t-1,a}$. $\hat{v}^n_{ta}$ is a sample estimate given the information $\hat{R}_{ta}(\omega^n)$, where ω^n is the sample at iteration n. If we are estimating a linear approximation, we would update our estimate of the slope using:

$$\bar{v}^n_{t-1,a} = (1-\alpha^n)\bar{v}^{n-1}_{t-1,a} + \alpha^n \hat{v}^n_{ta} \tag{18}$$

where $0 \leq \alpha^n \leq 1$ is the stepsize at iteration n. We note that $\hat{v}^n_{ta}$ is indexed by t (following our convention) because it is a sample realization using information from time t. $\bar{v}^n_{t-1,a}$ is indexed by $t-1$ because it is an approximation of an expectation at time $t-1$ (before the information during time interval t becomes available).

It is important to appreciate a critical difference between our updating scheme and traditional estimation methods in approximate dynamic programming. In the vast majority of applications, the parameter vector θ is estimated using observations of the value function. Using our notation, by iteration n we would have observed $\tilde{V}{}^{x,m}_{t-1}(R^{x,m}_{t-1}, \omega^m)$ for state $R^{x,m}_{t-1}$ for $m = 1, 2, \ldots, n$. Classical applications of approximate dynamic programming would use the sequence of observations of the pairs $(\tilde{V}{}^{x,m}_{t-1}, R^{x,m}_{t-1})$ to estimate the best value of the vector θ (or θ_t). That is, at each iteration we could solve:

$$\theta^n_t = \arg\max_{\theta} \sum_{m=1}^{n} (\bar{V}_t(R^{x,m}_t|\theta) - \tilde{V}{}^{x,m}_t)^2$$

Solving this nonlinear programming problem is impractical in many applications, but we can use recursive methods. But the major limitation is that we obtain a single observation $\tilde{V}{}^{x,n}_t$ at each iteration, from which we have to estimate a potentially very large vector θ_t. For our problem, we obtain a vector $\hat{v}^n_t = (\hat{v}^n_{ta})_{a\in\mathcal{A}}$ at each iteration. Furthermore, the updating method is quite simple, making the method computationally attractive for large-scale problems.

For many applications, linear (in the resource vector) approximations will not produce the best results. If we obtain an estimate $\bar{v}^n_{ta}$ that is too high, we may be more likely to create too many resources with attribute a. As $\bar{v}^n_{ta}$ bounces up and down from one iteration to the next, the system may swing wildly from too many resources (of this type) to too few.

If this behavior occurs, a nonlinear (piecewise linear) approximation may be much more stable. These are still quite easy to estimate. If we observe the resource vector $R^{x,n}_{t-1}$ and obtain a marginal value $\hat{v}^n_{ta}$, we can update the slope $\bar{v}_{t-1,ai}$ corresponding to the segment $i = R^{x,n}_{t-1}$.

$$\bar{v}^n_{t-1,ai} = (1-\alpha^n)\bar{v}^{n-1}_{t-1,ai} + \alpha^n \hat{v}^n_{tai} \tag{19}$$

TABLE I

PERCENTAGE OF OPTIMAL SOLUTION (FROM LP SOLVER) OBTAINED USING PIECEWISE LINEAR, SEPARABLE APPROXIMATIONS (FROM (7)).

	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%

One problem with this updating system is that we are depending on maintaining a sequence of value function approximations that are concave (convex if we are minimizing). Updating the single slope $\bar{v}^{n-1}_{t-1,ai}$ using the random outcome $\hat{v}^n_{tai}$ may easily produce a slope that violates concavity, where we would require that $\bar{v}^n_{t-1,a,i-1} \geq \bar{v}^n_{t-1,ai} \geq \bar{v}^n_{t-1,a,i+1}$. There are, however, several simple methods for maintaining concavity. Let $\hat{v}^n(\omega)$ be our sample estimate of a slope when the resource level $R^n(\omega) = r$, and let

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

We can regain concavity by applying the following simple procedure:

$$\bar{v}^{n+1}_r = \begin{cases} \tilde{v}^n_r & \text{if } R^n(\omega) = r \\ \tilde{v}^n_r & \text{if } R^n(\omega) = i < r \text{ and } \tilde{v}^n_r < \bar{v}^n_r \\ & \\ \tilde{v}^n_r & \text{if } R^n(\omega) = i > r \text{ and } \tilde{v}^n_r > \bar{v}^n_r \\ \bar{v}^n_r & \text{otherwise} \end{cases} \tag{20}$$

There are other methods (see, for example, references (5) and (6)), but this illustrates the basic idea of maintaining the basic structure of the approximation.

A major benefit of using piecewise linear functions is that we do not have to assume a functional form for the approximation. The downside would appear to be the need to estimate a large number of parameters (that is, every slope, as opposed to a few parameters of a polynomial approximation). However, using structure such as concavity dramatically improves the rate of convergence, since an observation of one slope may produce updates to other slopes.

IV. SOME EXPERIMENTAL RESULTS

Computational experiments have been very promising on certain classes of resource allocation problems. Optimal solutions are not available, so other strategies are needed to provide a measure of solution quality. One strategy is to solve a deterministic problem, since we can solve these problems using commercial linear programming solvers. Table I reports on a set of experiments involving the movement of transportation containers between 20, 40 and 80 locations, for problems with 15, 30 and 60 time periods. In all cases, using piecewise linear, separable approximations produced solutions that were virtually optimal.

The method has been applied to two-stage resource allocation problems. This is a special problem class where you allocate resources (often, you have to acquire them) and then live with the consequences. If the second stage is

TABLE II

THE AVERAGE OBJECTIVE FUNCTION VALUE NORMALIZED BY THE AVERAGE OF THE POSTERIOR BOUNDS FOR EACH SAMPLE REALIZATION FOR LINEAR AND PIECEWISE LINEAR, SEPARABLE APPROXIMATIONS, COMPARED TO A ROLLING HORIZON APPROXIMATION (FROM (8)).

Locations	Approx.	Mean	Std. dev.
10	Linear	84.55	2.339
	Separable Concave	95.18	2.409
	Rolling Horizon	91.45	2.348
20	Linear	80.52	2.463
	Separable Concave	95.48	2.153
	Rolling Horizon	88.91	1.930
40	Linear	74.13	1.816
	Separable Concave	92.21	0.465
	Rolling Horizon	86.89	0.772

separable (there is no substitution), then the piecewise linear approximations have been shown to be optimal, but it appears to provide near optimal solutions, with much faster rates of convergence than existing methods, even when the second stage is nonseparable (see references (6)).

We do not have optimal solutions for multistage stochastic resource allocation problems, but we can compare against classical rolling horizon experiments where decisions at time t use point forecasts of future events. After implementing decisions for time period t, we step forward in time, sample from the forecasts and then solve a new problem. Table IV summarizes the results for both linear and piecewise linear approximations, against the results from a rolling horizon heuristic. The results indicate that the linear approximation does not work well, but the separable, piecewise linear approximation works quite well.

This work suggests that these approximations can work well for certain problem classes. These tests have been done in the context of managing discrete resources (boxcars, containers, trucks, planes) used in transportation. In all these experiments, we are managing a resource (the equipment) to serve tasks (loads to be moved, passengers to be served, flights to be covered) which must be covered at a certain point in time.

V. RESEARCH CHALLENGES

Whenever an algorithmic strategy works well for a problem class, the natural question is the range of problems for which the strategy can be expected to work well. In this section, we touch on some complications that require additional research.

A. Complex attribute spaces

It is not unusual when modeling large numbers of resources to model them in a fairly simple way. Containers in freight transportation might be characterized by the type of container and its location. An electric power transformer might be described by the type of transformer, its location and its age.

In practice, even simple resources can become surprisingly complicated. A boxcar (used by railroads) might need the attributes of maintenance status, ownership and in-transit status (as well as location and type). People (pilots, equipment operators) and complex equipment (aircraft, locomotives) may require a dozen attributes or more. Very quickly, the attribute space $\mathcal{A}$ may grow too large to enumerate.

The results above assumed that we could enumerate the attribute space, obtaining a complete gradient, with a value $\hat{v}_{ta}$ for all $a \in \mathcal{A}$. This assumption is fairly critical. When the resources become too complicated, we encounter our own curse of dimensionality. In fact, it is useful to think of the attribute vector a as the state of a single resource. If this state space is too big, we have problems.

When we are managing large numbers of resources, it is natural to obtain the gradients $\hat{v}_{ta}$ whenever $R_{ta} > 0$. This is comparable to obtaining the value of being in a state for only the states we have visited. Overcoming this problem requires resorting to familiar techniques used in discrete dynamic programming such as aggregation. For example, let $G : \mathcal{A} \rightarrow \mathcal{A}^g$ be an aggregation function on the attribute space $\mathcal{A}$. Each time we find a marginal value $\hat{v}_{ta}$ for some a, we may update an aggregated estimate $\bar{v}^g_{ta}$ for $a \in \mathcal{A}^g$. We can then use an aggregated value function $\bar{V}^g(R) = \sum_{a \in \mathcal{A}^g} \bar{v}^g_{ta} R_{ta}$. We then face the typical problem of finding the best level of aggregation.

Our work has shown that there is no single best level of aggregation. The best level of aggregation depends largely on how many observations we have of each part of the attribute space. This depends on how many iterations we have run the algorithm, and we also have to recognize that some attributes are visited more often than others. Recognizing this, it makes sense to use a series of aggregation functions $\mathcal{G}$. Let w^g_a be the weight given to the g^{th} level of aggregation when we are estimating the value of attribute a. Our value function approximation would then be given by:

$$\bar{V}_t(R_t) = \sum_{g \in \mathcal{G}} \sum_{a \in \mathcal{A}^g} w^g_a \bar{v}^g_a$$

An effective method for computing the weights is as follows. Let $(s^2_a)^g$ be an estimate of the variance of $\bar{v}^g_a$. For higher levels of aggregation, we obtain more observations (which decreases $(s^2_a)^g$) but more greater errors due to the effects of aggregation (which increases $(s^2_a)^g$). Typically, this method puts higher weights on more aggregate levels initially, with increasing weights on the more disaggregate levels as the number of iterations increases. While there is modest overhead to compute $(s^2_a)^g$, the result is a smooth transition from higher to lower levels of aggregation, depending on how many times we sample an attribute a.

An indication of the effectiveness of this strategy is given in figure 1, which shows the objective function produced at each iteration using two levels of aggregation, and a weighted combination of the two.

B. Multiple resource classes

Up to now we have considered problems with a single *resource class*. Many problems involve managing a single type of resource characterized by an attribute space $\mathcal{A}$. More complex problems involve multiple classes: aircraft and pilots;

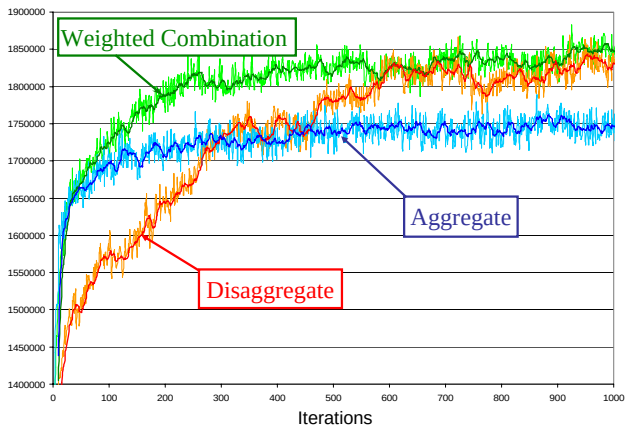

Fig. 1. Value of hierarchical learning with complex attributes

locomotives and trains; programmers and programming jobs; military aircraft and refueling tankers. In these problems, we have to manage two different classes of resources which interact with each other.

The simplest type of two class problem arises when resources are used to serve customer demands (jobs, tasks) which if unserved, are held for later. Known in some communities as backlogging, we have to model both the state of the resources, and the state of the unserved demands.

Such problems are characterized by a resource vector which might be given by $R_t = (R_t^{resource}, R_t^{demand})$. The question is: what type of value function approximation will be effective? We have been working with separable approximations (across the attribute space), but can we extend this to multiple resource classes:

$$\bar{V}_t = \sum_{a \in \mathcal{A}^{resource}} \bar{V}_{ta}^{resource}(R_{ta}^{resource}) + \sum_{a \in \mathcal{A}^{demand}} \bar{V}_{ta}^{demand}(R_{ta}^{demand})$$

There is limited evidence that indicates that this can work ((9)), but intuition suggests that such a strategy may have problems. The value of a particular number of resources depends, naturally, on the number of demands to be served. This simple observation argues for a nonseparable approximation, which can be quite hard to build if we are managing discrete resources. It is very useful having piecewise linear approximations defined on the integer lattice. This is much easier to do for a scalar function.

It is possible, of course, to propose an approximation by suggesting a series of basis functions which might include products between the resource classes as well as separable functions of the resource classes themselves. For the case of discrete resources, such approximations produce difficult integer nonlinear programming problems which are likely to be intractable for large problems. There may be opportunities to take advantage of techniques from stochastic integer programming (see, for example, references (10)), but this is an area for future research.

VI. Conclusions

The problem of managing multiple discrete resources can dramatically increase the size of the state space. We describe an algorithmic strategy based on the use of continuous, separable value function approximations, along with a form of the optimality equations that depends on using a post-decision state variable, which is proving useful on particular classes of problems.

Acknowledgment

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

References

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

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

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

[4] D. P. Bertsekas, V. S. Borkar, and A. Nedich, "Improved temporal difference methods with linear function approximation," in *Handbook of Learning and Approximate Dynamic Programming*, J. Si, A. G. Barto, W. B. Powell, and D. W. II, Eds. IEEE Press, 2004.

[5] G. A. Godfrey and W. B. Powell, "An adaptive, distribution-free approximation for the newsvendor problem with censored demands, with applications to inventory and distribution problems," *Management Science*, vol. 47, no. 8, pp. 1101–1112, 2001.

[6] W. B. Powell, A. Ruszczyński, and H. Topaloglu, "Learning algorithms for separable approximations of stochastic optimization problems," *Mathematics of Operations Research*, vol. 29, no. 4, pp. 814–836, 2004.

[7] G. Godfrey and W. B. Powell, "An adaptive, dynamic programming algorithm for stochastic resource allocation problems I: Single period travel times," *Transportation Science*, vol. 36, no. 1, pp. 21–39, 2002.

[8] H. Topaloglu and W. B. Powell, "Dynamic programming approximations for stochastic, time-staged integer multicommodity flow problems," *Informs Journal on Computing*, to appear.

[9] M. Spivey and W. B. Powell, "The dynamic assignment problem," *Transportation Science*, vol. 38, no. 4, pp. 399–419, 2004.

[10] W. K. K. Haneveld and M. H. van der Vlerk, "Stochastic integer programming:general models and algorithms," *Annals of Operations Research*, vol. 85, pp. 39–57, 1999.