Approximate Dynamic Programming - I: Modeling

Warren B. Powell

December 7, 2009

Abstract

The first step in solving a stochastic optimization problem is providing a mathematical model. How the problem is modeled can impact the solution strategy. In this chapter, we provide a flexible modeling framework that uses a classic control-theoretic framework, avoiding devices such as one-step transition matrices. We describe the five fundamental elements of any stochastic, dynamic program. Different notational conventions are introduced, and the types of policies that can be used to guide decisions are described in detail. This discussion puts approximate dynamic programming in the context of a variety of other algorithmic strategies by using the modeling framework to describe a wide range of policies. A brief discussion of model-free programming is also provided.

1 Introduction

Stochastic optimization problems pose unique challenges in how they are represented mathematically. These problems arise in a number of different communities, often in the context of problems which introduce specific computational characteristics. As a result, a number of contrasting notational styles have evolved which complicate our ability to communicate research across communities. This is particularly problematic in the general area of multistage, stochastic optimization problems, where different communities have made significant algorithmic contributions which have applications to a wide variety of problems.

The range of problems that can be modeled as stochastic, dynamic optimization problems is vast. Examples of major problem classes include:

- Optimization over stochastic graphs - This is a fundamental problem class that addresses the problem of managing a single entity in the presence of different forms of uncertainty with finite actions.
- Dynamic resource allocation problems - These include scheduling people and machines, routing vehicles, managing inventories, and investing in new facilities and technologies. These problems arise in supply chain management, personnel management, health care, military operations, agriculture and energy.
- Demand management - These problems include booking strategies for airlines, hotels, hospitals, vendor-managed inventories, and incentives to control the demand for energy.
- Management of financial portfolios - How should a portfolio be spread over different investments to strike a balance between risk and return?
- R & D portfolio problems - How should research and development portfolios be managed to reach specific technological goals? What investment strategy should we pursue to ensure that we will meet government targets for renewable energy in 30 years? These decisions need to be made in the presence of uncertainty about prices, climate, technology and government policy.
- Pricing problems - How should products and services be priced to maximize total revenue?
- Engineering control problems - How much CO2 should we release into the atmosphere? What time window should you commit to for providing service? At what speed should you fly your aircraft?
- Sensor management problems - We would like to manage a team of technicians collecting information about the presence of disease in the population, the concentration of pollution or radiation in the atmosphere, or the concentration of pollutants in the water.

These problems are hardly exhaustive, but hint at the range of applications and types of complexities that we might encounter. In all of these problems, we face the challenge of making decisions sequentially, in that we make a decision, and then observe information that we did not know when we made the first decision. We then get to make another decision, after which we see more information. The goal is to make decisions over time that achieve some objective.

There are several ways to model these problems, and different communities have evolved modeling and algorithmic strategies to deal with specific problem classes. For example, the simulation community typically uses myopic policies (rules that do not directly consider the impact of decisions now on the future) which might depend on one or more tunable parameters. For example, an (q, Q) inventory policy orders new product if the inventory falls below q, and places an order to bring the total inventory up to Q. In this case, q and Q are tunable parameters which can be optimized to find the best policy, indirectly taking into account the impact of decisions now on the future.

The Markov decision process community assumes that we can represent our system as being in a state s at time t. If we choose action a, then we let $p(s'|s, a)$ be the probability that we then land in state s'. If $C(s, a)$ is the contribution (reward) we earn if we choose action a when in state s, then we can find the best action by solving Bellman's optimality equation, given by

$$V(s) = \max_a \big(C(s, a) + \gamma \sum_{s'} p(s'|s, a)V(s')\big), \tag{1}$$

where γ is a discount factor. We are assuming that we are maximizing total discounted contributions over an infinite horizon. The challenge is computing the value $V(s)$ for each (discrete) state s. There are powerful algorithms for solving this problem, but they require enumerating the set of potential states. While there are many problems that can be solved with this strategy, the method breaks down when s consists of a vector of elements. This produces the well-known curse of dimensionality of dynamic programming.

Approximate dynamic programming (ADP) is both a modeling and algorithmic framework for solving stochastic optimization problems. Most of the literature has focused on the problem of approximating $V(s)$ to overcome the problem of multidimensional state variables. In addition to the problem of multidimensional state variables, there are many problems with multidimensional random variables, and multidimensional decision variables (most commonly referred to as actions in the dynamic programming community, or controls in the engineering literature). These three challenges make up what have been called the three curses of dimensionality.

It is important in any presentation on dynamic programming to acknowledge the different communities that have contributed to the field. The challenge of making good decisions over time in the presence of uncertainty arises in a number of fields, and as a result it is not surprising to see similar ideas being developed under different notational systems and different vocabularies. These communities include:

- Discrete Markov decision processes (MDP's) - This covers research in computer science as well as the MDP community in operations research. These problems are typically characterized by discrete states (with possibly many states), and discrete actions, but typically not very many actions.
- Control theory - These communities include engineering in the physical sciences and economics. Problems are often modeled in continuous time, with decision variables (controls) that are typically continuous and low-dimensional (e.g. one to a dozen dimensions). Randomness often arises in the form of measurement noise.
- Stochastic programming - This community deals with vector-valued (and often high- dimensional) decision vectors and general forms of uncertainty which are represented using scenario trees. This community typically does not use Bellman's optimality equation as an algorithmic device.

- Simulation optimization - The simulation community generally uses myopic policies to make decisions over time, but these policies may be governed by a vector of tunable parameters that can be optimized. This community also does not use Bellman's equation to guide decisions, but there are close parallels between the problem of optimizing policies in simulation, and policy optimization in dynamic programming.

Although the roots of approximate dynamic programming can be traced to early work by Bellman (see, for example, Bellman and Kalaba (1959)), the ideas evolved independently within different fields, notably the early work on training computers to play games (Samuel (1959, 1967)), and the work in control theory (Werbos (1974), Werbos (1989), White and Sofge (1992)). The work in computer science evolved under the name "reinforcement learning," where the first published use of this term is Minsky (1961) (the roots of this work can be found in Minsky's Ph.D. dissertation Minsky (1954); see also Mendel and McLaren (1970)). Reinforcement learning as a field did not really emerge until the 1980's with Barto et al. (1981), followed by numerous contributions by Sutton and Barto through the 1980's, eventually leading to their ground breaking book Sutton and Barto (1998). Work in control theory took place under a variety of names, including reinforcement learning, adaptive dynamic programming and (later) approximate dynamic programming, with important early contributions by Paul Werbos (see Werbos (1974), Werbos (1989), White and Sofge (1992) and Si et al. (2004)). The seminal book Bertsekas and Tsitsiklis (1996) introduced the name "neuro-dynamic programming," but it appears that this term is being replaced with approximate dynamic programming (see, for example, chapter 6 of Bertsekas (2007)).

While ADP in its various forms really accelerated in the 1990's in computer science and control theory, there was relatively little attention given to ADP in the operations research community until after year 2000. One of the earliest papers in the operations research literature to explicitly use the term approximate dynamic programming is Bertsimas and Demir (2002), although others have done similar work under different names such as adaptive dynamic programming (see, for example, Powell et al. (2001), Godfrey and Powell (2002), Papadaki and Powell (2003)). Methods for handling vector-valued decision variables in a formal way using the language of dynamic programming appear to have emerged quite late (see in particular Powell and Van Roy (2004)), although other authors have used specialized techniques from math programming to solve multistage stochastic optimization problems. Pereira and Pinto (1991) in particular introduces the idea of using Benders cuts to overcome the curse of dimensionality in dynamic programming (see also Powell (2007), chapter 11). This idea has enjoyed a substantial literature (see Birge and Louveaux (1997), Higle and Sen (1996)), but these have evolved independently of the approximate dynamic programming community.

From this discussion, we feel that any discussion of approximate dynamic programming has to acknowledge the fundamental contributions made within computer science (under the umbrella of reinforcement learning) and control theory. The one dimension that these communities largely ignored was problems which involved high-dimensional decision variables, which are common in operations research. The first book to bridge the gap with mainstream operations research in a thorough way did not appear until Powell (2007).

2 Modeling a stochastic optimization problem

Before we can solve a problem, we have to model it. In this section, we review the five fundamental dimensions of a stochastic, dynamic systems which include 1) states, 2) actions/decisions/controls,

3) exogenous information/random variables, 4) transition function, and 5) objective function. We use this presentation to review the different notational systems used by different communities.

2.1 States

It is with some surprise that we have found that few authors attempt to actually define a state variable. Powell (2007) offers the following definition:

Definition 2.1 *A* **state variable** *is the minimally dimensioned function of history that is necessary and sufficient to compute the decision function, the transition function, and the contribution function.*

This definition is familiar to researchers in control theory (in particular, in electrical engineering). The modifier "minimally dimensioned" is intended to restrict the state variable to all the information needed, but only the information needed, so that it is as compact as possible.

The Markov decision process framework, used in both computer science and operations research, uses s for state, or S_t for the random variable describing the state at time t. Control theorists use x. Most applications involve modeling a physical state (the status of a piece of equipment, the amount of products in different inventories), but many problems require modeling an information state (information used to make a decision), and for some applications, a belief state (when we are unsure about the actual state of our system).

2.2 Decisions/actions/controls

We might refer to action a (Markov decision processes), control u or decision x. While it is easy to view these as different variable names for the same quantity, it is also the case that these communities tend to work on different types of problems. Although exceptions abound, in most cases a refers to a relatively small (that is, easy to enumerate) set of discrete actions; the control u is typically a low-dimensional (e.g. one to ten) continuous vector (density, pressure, velocity, acceleration, price); and the decision vector x in operations research is often very high-dimensional, with hundreds to tens of thousands of dimensions.

We defer until later the problem of determining how to make a decision, other than to say that we will ultimately look for a *policy* π, which is a rule (or function) for determining a decision using the information in the state variable S_t. We might use π to represent this rule, or a function $A^\pi(S_t)$ to determine the action a_t or $X^\pi(S_t)$ to determine the decision x_t. We let Π be the set of all possible policies (or functions), which takes on different meanings as we give a policy structure in a specific setting.

2.3 Exogenous information

We are typically interested in problems that are driven by some sort of exogenous information process. This might come from a physical system (observed prices or rainfall) or a probability distribution. These are modeled as random variables, but there is an important distinction between whether the underlying probability distribution is known or not.

While communities have standard notation for states and actions, we are unaware of any standard notation for exogenous information. The MDP community typically does not explicitly model exogenous information, preferring the more compact representation of the one-step transition function $p(s'|s, a)$. In control theory, we often see w_t for random information. Given the preference of the applied probability community to use capital letters for random variables, we use W_t as our generic notation for random information.

A separate modeling issue for discrete time models is the modeling of time. In a continuous time model, W_t would represent information arriving between t and $t + dt$. For discrete time problems, there are many authors who would let W_t be new information (about rainfall, changes in prices, new demands) arriving between t and $t + 1$, but this means that at time t, W_t is random. We prefer the convention, widely used in the applied probability community, that W_t represents information arriving between $t - 1$ and t. With this convention, any variable indexed by t is known at time t.

We let ω represent a sample path $(W_1, W_2, \ldots)$ where $\omega \in \Omega$. To finish the formalism, we let $\mathcal{F}$ be the sigma-algebra on Ω (the set of events), and let $\mathcal{P}$ be a probability measure on $(\Omega, \mathcal{F})$. Finally, because information evolves, over time, we let $\mathcal{F}_t$ be the sigma-algebra generated by the variables $(W_1, \ldots, W_t)$, which implies that $\mathcal{F}_t \subseteq \mathcal{F}_{t+1}$ is a sequence of filtrations. We can use this notation to express the dependence of state variables and decisions on information that has arrived prior to time t.

There are instances where it is useful to provide an explicit model of the history. For this purpose, we can define

$$\begin{aligned}
H_t &= \text{The history of the process, consisting of all the information known} \\
&\quad\ \text{through time } t, \\
&= (W_1, W_2, \ldots, W_t), \\
\mathcal{H}_t &= \text{The set of all possible histories through time } t, \\
&= \{H_t(\omega) | \omega \in \Omega\}, \\
h_t &= \text{A sample realization of a history}, \\
&= H_t(\omega), \\
\Omega(h_t) &= \{\omega \in \Omega | H_t(\omega) = h_t\}.
\end{aligned}$$

In the stochastic programming community, it is common to model uncertainty through scenario trees, which represents the branching of outcomes as new information becomes available. Imagine all the outcomes $\omega \in \Omega(h_t)$ which correspond to a common history h_t at time t. We can model this juncture as a node $n \in \mathcal{N}$, where each node n captures a particular history at a point in time. All the outcomes that meet at node n follow a common path (corresponding to the history h_t). Rather than use an explicit model of time, the modeling of scenario trees typically refers to predecessor nodes and successor nodes. A decision made at node n of the tree has to depend on the information up to that juncture. This representation does not use the concept of a state variable, where outcomes with different histories can lead to the same state.

It is interesting to contrast the use of scenario trees in stochastic programming with state variables in dynamic programming. A node in a scenario tree corresponds to an entire history. Not surprisingly, scenario trees grow exponentially in size as the number of time periods increases. For this reason, there is a literature addressing the problem of generating scenario trees which capture desirable properties with a minimum number of outcomes (see Dupačová et al. (2000), Hø yland and

Wallace (2001), Heitsch and Romisch (2009)). It is perhaps curious that the stochastic programming community, which focuses primarily on problems with multidimensional decisions x_t, views dynamic programming as a method that is limited to small problems (due to the "curse of dimensionality") when in fact scenario trees suffer from a similar curse of dimensionality when representing exogenous information processes.

In practice, scenario trees are used most often when the history of a process plays an important role. Although the history can be easily added to a state variable, the result is an extremely large state space where there may be a unique state for each sample path (which shares a common history). We note that scenario trees are generated prior to solving the problem, which means that this method of modeling uncertainty is unable to handle problems where the exogenous outcome depends on a prior decision. For example, we may be modeling random prices to determine the value of an asset. If we sell more, the prices may drop. For such problems, we cannot generate scenario trees in advance.

2.4 The transition function

There are different styles for modeling how the system evolves over time. The convention in operations research is to use systems of equations, such as

$$A_t x_t + B_{t-1} x_{t-1} = b_t. \tag{2}$$

In the MDP community, the evolution of the system is described using the one-step transition matrix $p(s'|s,a)$.

In the control theory community, it is common to define a function that maps state, action, new information to new state, as in:

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

The function $S^M(\cdot)$ goes under many names: "plant model" (literally, the model of a physical production plant), "plant equation," "law of motion," "transfer function," "system dynamics," "system model," "transition law," and "transition function." We use "transition function," but adopt notation that captures the widely used term "system model." Transition functions are typically straightforward to specify (with exceptions as we note below), although in many engineering applications, they can be quite complex. For this reason, it is common practice to specify the existence of a transition function when modeling a problem without actually writing out the details. This contrasts sharply with writing out systems of linear equations such as equation (2), where all the details of the transition are fully specified.

It is often overlooked that the one-step transition matrix is actually an expectation, since it can be derived directly from the transition function as follows:

$$\begin{aligned}
\mathbb{P}(s'|s,a) &= \mathbb{E}\{1_{\{s'=S^M(S_t,a,W_{t+1})\}}|S_t = s\} \\
&= \sum_{\omega_{t+1}\in\Omega_{t+1}} \mathbb{P}(W_{t+1} = \omega_{t+1}) 1_{\{s'=S^M(s,a,W_{t+1})\}}.
\end{aligned}$$

The problem with one-step transition matrices is that they tend to be extremely large, measuring the number of states times the number of states times the number of actions. One-step transition matrices have enjoyed a rich history in the literature for Markov decision processes, where they have facilitated an elegant theory. But in practice, it is only an extremely narrow class of problems where they can actually be computed.

2.5 Objective function

The final dimension of our model is the objective function. We might minimize a cost or maximize a contribution (or reward). Assuming that we are maximizing, we define

$C(S_t, a_t) =$ Contribution received for being in state S_t and taking action a_t.

The contribution might be a random variable, which we would then write as $C(S_t, a_t, W_{t+1})$. In many applications, we observe the next state S_{t+1} but do not explicitly observe W_{t+1}, in which case we may write the contribution as $C(S_t, a_t, S_{t+1})$. In either case, $C(S_t, a_t)$ would be the expected contribution. The most common assumption is that we are maximizing total discounted rewards over a finite or infinite horizon, which we would write as

$$F^\pi(S_0) = \mathbb{E}\left\{\sum_{t=0}^{T} \gamma^t C(S_t, A^\pi(S_t))\right\}, \tag{3}$$

where γ is a discount factor. It is very common to let $T \to \infty$ and solve for a steady state policy, but for other problems, solving undiscounted, finite horizon problems is the standard model. We note that we are assuming that our objective function can be written in the form of additive rewards.

We let $A^\pi(S_t)$ be a function, parameterized in some way by $\pi \in \Pi$, that determines the action a_t given the information in the state S_t. Our goal is to find the best policy, which means solving

$$\max_{\pi\in\Pi} \mathbb{E}\left\{\sum_{t=0}^{T} \gamma^t C(S_t, A^\pi(S_t))\right\}. \tag{4}$$

Solving this optimization problem directly, even for very simple problems, is computationally intractable. The breakthrough of dynamic programming is realizing that this problem can be solved using Bellman's optimality equation (see Puterman (1994) for a modern and thorough discussion of this field), which can be written

$$\begin{aligned} V_t(S_t) &= \max_a \big(C(S_t, a) + \gamma \sum_{s'} p(s'|S_t, a) V_{t+1}(s')\big), & (5)\\ &= \max_a \big(C(S_t, a) + \gamma \mathbb{E}\{V_{t+1}(S_{t+1})|S_t\}\big), & (6) \end{aligned}$$

where $S_{t+1} = S^M(S_t, a, W_{t+1})$. For steady state problems, we let $V_t(S) = V_{t+1}(S) = V(S)$. If states are discrete (and there are not too many of them), the number of actions is not too large, and the expectation is easy to compute, then equation (5) or (6) can be used to find the best action a for each state S, which gives us a lookup-table representation of a policy. Unfortunately, only a small number of toy problems meet these criteria, which is what leads us to the field of approximate dynamic programming.

3 Major problem classes

Approximate dynamic programming arises because of the computational difficulties in solving Bellman's equation. Now that we have our modeling framework in place, we can discuss more precisely about the nature of these complexities.

There are three computational challenges that arise in the solution of Bellman's equation:

1) Finding the value function $V(S)$ (or $V_t(S_t)$ for finite-horizon problems).
2) Computing the expectation.
3) Finding the best action.

The nature of these challenges arises from the characteristics of three variables: the state variable S_t, the exogenous information variable W_t, and the action a_t/control u_t/decision x_t.

State variables can typically be divided between whether the state space is a) discrete and easy to enumerate (up to thousands of states, but not millions), b) scalar and continuous, or c) a vector (whether it is discrete or continuous). Often, cases (b) and (c) are equivalent from a computational perspective, but scalar, continuous variables tend to offer special structure. The third case spans discrete vectors (how many cars of each model are in inventory), continuous vectors (how much money is invested in each investment choice), vectors of categorical attributes, and of course a mixture of all of these. There are many applications where the state variable is complicated by the need to retain some portion of the history of the process. When this happens, a common modeling strategy is to use scenario trees.

For the random vector W_t, we are primarily interested in whether we can compute the expectation in Bellman's equation. Vectors of random variables may be easy if they are independent, and of course the problem is easiest when there are no autocorrelations linking observations over time. It is possible the random information is simple, but we do not know the probability distribution. For example, the random variable may simply capture whether a person accepts a bid in an auction or not; the random variable is Bernoulli, but we do not know the probability distribution describing the person's behavior. It is also important to separate problems where W_t is independent of all prior history; problems where W_t depends on the state S_t; and problems where W_t depends on both the state S_t and action x_t.

For decisions, there may be a small number of discrete actions a, a single scalar decision, low-dimensional continuous vectors u, or high-dimensional continuous or discrete vectors x. For vector-valued actions, we typically need some sort of search algorithm such as a linear program or genetic algorithm to find x. The choice of algorithmic search strategy can have an impact on how we represent future events when making a decision.

The default way to handle any dynamic program is to discretize all the states and actions, and assume that we can compute expectations. When the number of states and actions is small enough to enumerate when discretized, and when we can compute expectations, we typically can solve Bellman's equation (1) exactly using classical techniques (see, e.g. Puterman (1994)). If the state variable is a vector, discretizing it can produce an exponentially large number of states, a problem that is routinely referred to as the "curse of dimensionality" (a term coined by Bellman). The same problem arises with the information variable W and the action $a/u/x$. However, there are problems where W may be continuous, but where the expectation is still easy. There are other problems where the information may be fairly simple (e.g. the behavior of an opponent or the price of a stock), but where we do not know the distribution so we cannot compute the expectation. Finally, there are problems where the decision variable is a vector, which makes it impossible to enumerate all the actions. When all three of these problems arise (vector-valued states, uncomputable expectation and vector-valued decisions), we say that we have three curses of dimensionality.

4 Types of policies

The challenge of dynamic programming is finding a good rule for making decisions given a state. We refer to this rule as a *policy*. Policies come in a number of forms, and the precise form can play a major role in the design of an algorithm for finding a good policy. A policy is often denoted as π, which is a generic mapping from a state to an action. It is convenient to emphasize that this is really a function. If the action is a, we might designate the function as $A^\pi(S)$ for the action that we would take if we are in state S. If your action is x, we could use $X^\pi(S)$. We design our space of possible policies using $\pi \in \Pi$, but what this means computationally is very dependent on the nature of the policy (or decision function). Examples of policies include:

1) Lookup tables - For a discrete state s, $A^\pi(s)$ is the discrete action we should take. If there are 100 states and 10 actions per state, our policy space would have 1,000 parameters.

2) Parameterized myopic policies - Let S_t be the amount of inventory on hand at time t. A reorder policy might be to order $X^\pi(S_t) = Q - S_t$ if $S_t < q$ and 0 otherwise. This policy is parameterized by q and Q, so we might say $\pi = (q, Q)$. The set of all possible policies is the set of potential values for q and Q.

3) Statistical models - When controlling energy commitments from a wind farm, let W_t be the wind speed. We have to decide how much energy to commit to the grid, which we might model using $x_t = \theta_0 + \theta_1 W_t + \theta_2 W_t^2 + \theta_3 W_t^3$. This regression function is a policy parameterized by $(\theta_0, \theta_1, \theta_2, \theta_3)$. Within this category, we would include training a neural network to represent a policy, a strategy that is common in the neural network community (see Haykin (1999), chapter 12).

4) Myopic optimization models - Let $C(S_t, x_t)$ be the contribution earned by using decision x when we are in state S. An example might be a resource allocation problem where we are allocating people, products or machinery to different tasks or demands. S_t captures the current status of our resources and tasks, and x_t is our vector of decisions of who gets assigned to what. We could solve our problem using

$$X^\pi(S_t) = \arg\max_{x \in \mathcal{X}} C(S_t, x).$$

This means finding the best assignment of resources now, without regard to the impact of these decisions on the future. We note that solving this problem may involve using a solver for linear, nonlinear or integer programs, or using a heuristic search algorithm such as tabu search or genetic algorithms.

5) Tree search - For problems with typically small action spaces, it is possible to estimate the value of a particular state by enumerating all the actions and subsequent states that result

from reaching a particular state. These methods are widely used in the design of algorithms for playing games (see Pearl (1984)).

6) Roll-out heuristics - When it is not possible to enumerate all the actions out of a state (as in tree search), it may be the case that we have access to a reasonable (but suboptimal) policy. A roll-out heuristic evaluates the value of a particular state s' (that we might reach from state s using a potential action a) by following this policy starting from s' for a specified number of iterations. The value of this simulated sample path can be used to approximate the value of reaching state s'. See Bertsekas and Castanon (1999) for a more in-depth discussion of this strategy.

7) Rolling horizon policies (deterministic) - We could also optimize over a planning horizon T using

$$U^\pi(S_t) = \arg\max_{u\in\mathcal{U}} \sum_{t'=t}^{t+T} C(S_{t'}, u_{t'}),$$

where point forecasts are used to make decisions over the horizon $t, \ldots, t+T$. We only use u_t as our decision to implement right now. This strategy is also known as a receding horizon policy, or, in the control theory community, model-predictive control. Rolling horizon policies are mathematically equivalent to tree search, but are normally written in the context of multidimensional decision/control problems where a solver of some sort is used to solve the optimization problem.

8) Rolling horizon policies (stochastic) - Classical rolling horizon policies use a point forecast of the future, but it is possible to use a stochastic model which captures uncertainty in potential future events. This strategy is most popular in the stochastic programming community which represents possible outcomes of future events as scenarios in a set Ω_t. An outcome $\omega \in \Omega_t$ would be viewed as a set of potential events over time periods $t, t+1, \ldots, t+T$.

$$X^\pi(S_t) = \arg\max_{x_t\in\mathcal{X}} C(S_t, x_t) + \sum_{\omega\in\Omega_t} p(\omega) \sum_{t'=t+1}^{t+T} C(S_{t'}(\omega), x_{t'}(\omega)).$$

These problems have to be solved subject to constraints (known as "nonanticipativity constraints" in the stochastic programming community) which ensure that a decision $x_{t'}$ does not see information that becomes available at time periods later than t'.

9) Value function approximations - In this strategy, we use an approximation of the value function in Bellman's equation, where we would make a decision by solving

$$A^\pi(S_t) = \arg\max_a \big(C(S_t, a) + \gamma \mathbb{E}\{\bar{V}_{t+1}(S^M(S_t, a, W_{t+1}))|S_t\}\big). \tag{7}$$

where $\bar{V}(S)$ is an approximation of the value of being in state S. The space of policies is the space of potential value function approximations.

Lookup tables are easy to visualize but require enumerating states and actions, so this is precisely the type of policy that is sensitive to curse of dimensionality issues. Finding the best parameters in a parameterized policy is a topic that has been widely studied in the literature known as simulation optimization (see, for example, Nelson et al. (2001), Fu et al. (2005), Kim and Nelson (2006), Chang et al. (2007)) where the techniques of stochastic search are used (see Spall (2003)). Rolling horizon policies have been viewed as a form of ADP (see Bertsekas (2005) for a discussion of the relationship between ADP and model-predictive control), since they are in the same mathematical class as tree-search algorithms and roll-out policies (Bertsekas and Castanon (1999)).

Approximate dynamic programming arises primarily when we are looking for a value function approximation that determines a decision using equations such as equation (7). However, finding the best statistical model (policies of type 3 above) is also an important strategy in the ADP literature, where it is referred to as approximate policy optimization. This is particularly common in the control theory community where a policy is a neural network (literally, a statistical model). It can be argued that finding the best regression function (type 3) and finding the best value function (type 9) are mathematically equivalent (both produce functions that are determined by regression methods), but the computational issues are different. The problem of finding the best approximation of a policy is closest to policy iteration of dynamic programming, while finding the best value function approximation is closest to value iteration.

5 Model-free dynamic programming

A topic that is very popular in computer science (in the reinforcement learning community) and engineering is a problem class that is referred to as "model free." These applications arise in the context of more complex applications, but the term "model free" can take on different meetings in different settings. In a nutshell, model-free dynamic programming arises when we cannot compute

$$a_t^n \quad = \quad \max_a \left(C(S_t^n, a) + \gamma \mathbb{E} V_{t+1}(S^M(S_t^n, a_t, W_{t+1}(\omega^n))\right). \tag{8}$$

There are three calculations implied in the solution of equation (8):

a) Computing $S_{t+1}^n = S^M(S_t^n, a_t, W_{t+1}(\omega^n))$ using the transition function,

b) computing the expectation and

c) computing the contribution function $C(S_t^n, a)$.

There are many applications where we cannot compute some combination of these calculations. The most common is problems where we do not have an explicit transition function. Given the fact that many refer to this as the model, the lack of a model (transition function) resulted in algorithmic strategies which address this dimension as model-free dynamic programming. These techniques apply equally to problems where we cannot compute the expectation, which can easily arise because observations are from an exogenous process where we do not know the underlying probability distribution. There are also problems where we do not have an explicit contribution (or reward or utility) function. These can arise when we are trying to mimic a human making decisions, where we do not know the precise utility function that guides the human.

The reinforcement learning community often requires a model-free framework since this community is frequently working on problems that involve mimicing human behavior. Model-free dynamic programming is so common, in fact, that authors feel that they have to explicitly state when an algorithm is model-based. The control theory community often encounters model-free applications when the physics of a particular problem (e.g. modeling a chemical plant) is simply too complex to represent as a mathematical model.

6 Closing remarks

The goal of this chapter was to outline a general strategy for modeling stochastic, dynamic problems. Designing effective policies is a difficult challenge which requires taking advantage of the nature of a particular problem. Approximate dynamic programming offers very general algorithmic framework for solving these problems. An introduction to this approach is given in Powell (2010) in this volume.

References

Barto, A., Sutton, R. and Brouwer, P. (1981), ‘Associative search network: A reinforcement learning associative memory’, *Biological cybernetics* **40**, 201–211.

Bellman, R. and Kalaba, R. (1959), ‘On adaptive control processes’, *IRE Transactions on Automatic Control* **4**, 19.

Bertsekas, D. (2005), ‘Dynamic programming and suboptimal control: A survey from ADP to MPC’, *European Journal of Control* **11**, 310–334.

Bertsekas, D. P. (2007), *Dynamic Programming and Optimal Control, Vol. II*, Athena Scientific, Belmont, MA.

Bertsekas, D. P. and Castanon, D. A. (1999), ‘Rollout Algorithms for Stochastic Scheduling Problems’, *J. Heuristics* **5**, 89–108.

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

Bertsimas, D. and Demir, R. (2002), ‘An approximate dynamic programming approach to multidimensional knapsack problems’, *Management Science* **48**, 550–565.

Birge, J. R. and Louveaux, F. (1997), *Introduction to Stochastic Programming*, Springer Verlag, New York.

Chang, H. S., Fu, M. C., Hu, J. and Marcus, S. I. (2007), *Simulation-Based Algorithms for Markov Decision Processes*, Springer, Berlin.

Dupaçová, J., Consigli, G. and Wallace, S. W. (2000), ‘Scenarios for multistage stochastic programs’, *Annals of Operations Research* **100**(1), 25–53.

Fu, M., Glover, F. and April, J. (2005), ‘Simulation optimization: a review, new developments, and applications’, *Proceedings of the 37th conference on Winter simulation* pp. 83—-95.

Godfrey, G. and Powell, W. (2002), ‘An adaptive dynamic programming algorithm for dynamic fleet management, I: Single period travel times’, *Transportation Science* **36**, 21–39.

Haykin, S. (1999), *Neural Networks: A Comprehensive Foundation*, Prentice Hall.

Heitsch, H. and Romisch, W. (2009), 'Scenario tree modeling for multistage stochastic programs', *Mathematical Programming* **118**(2), 371–406.

Higle, J. L. and Sen, S. (1996), *Stochastic Decomposition: A Statistical Method for Large Scale Stochastic Linear Programming*, Kluwer Academic Publishers.

Hø yland, K. and Wallace, S. W. (2001), 'Generating scenario trees for multistage decision problems', *Management Science* pp. 295–307.

Kim, S. H. and Nelson, B. L. (2006), *Selecting the best system*, Elsevier, chapter 17.

Mendel, J. M. and McLaren, R. W. (1970), *Reinforcement learning control and pattern recognition systems*, Vol. 66, Academic Press, New York, pp. 287–318.

Minsky, M. L. (1954), Theory of neural-analog reinforcement systems and its application to the brain-model problem, PhD thesis.

Minsky, M. L. (1961), 'Steps Toward Artificial Intelligence', *Proceedings of the Institute of Radio Engineers* **49**, 8–30.

Nelson, B. L., Swann, J., Goldsman, D. and Song, W. (2001), 'Simple procedures for selecting the best simulated system when the number of alternatives is large', *Operations Research* **49**, 950–963.

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

Pearl, J. (1984), *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley.

Pereira, M. V. F. and Pinto, L. M. V. G. (1991), 'Multistage stochastic optimization applied to energy planning', *Mathmatical Programming* **52**, 359–375.

Powell, W. B. (2007), *Approximate Dynamic Programming: Solving the curses of dimensionality*, John Wiley & Sons, Hoboken, NJ.

Powell, W. B. (2010), *Approximate Dynamic Programming - II: Algorithms*, John Wiley and Sons, chapter ???, p. ???

Powell, W. B., Shapiro, J. A. and Simao, H. P. (2001), *A representational paradigm for dynamic resource transformation problems, in R*, J. C. Baltzer AG, pp. 231–279.

Powell, W. B. and Van Roy, B. (2004), Approximate Dynamic Programming for High Dimensional Resource Allocation Problems, *in* J. Si, A. G. Barto, W. B. Powell and D. W. II, eds, 'Handbook of Learning and Approximate Dynamic Programming', IEEE Press, New York.

Puterman, M. L. (1994), *Markov Decision Processes*, John Wiley & Sons, Hoboken, NJ.

Samuel, A. L. (1959), 'Some studies in machine learning using the game of checkers', *IBM Journal of Research and Development* **3**, 211—-229.

Samuel, A. L. (1967), 'Some studies in machine learning using the game of checkers II - recent progress', *IBM J. Res. Develop* **11**, 601—-617.

Si, J., Barto, A. G., Powell, W. B. and Wunsch, D. (2004), 'Handbook of learning and approximate dynamic programming', *Wiley-IEEE Press* .

Spall, J. C. (2003), *Introduction to Stochastic Search and Optimization: Estimation, Simulation and Control*, John Wiley & Sons, Hoboken, NJ.

Sutton, R. and Barto, A. (1998), *Reinforcement Learning*, Vol. 35, MIT Press, Cambridge, MA.

Werbos, P. J. (1974), Beyond regression: new tools for prediction and analysis in the behavioral sciences, PhD thesis.

Werbos, P. J. (1989), 'Backpropagation and neurocontrol: A review and prospectus', *Neural Networks* pp. 209–216.

White, D. A. and Sofge, D. A. (1992), *Handbook of intelligent control: Neural, fuzzy, and adaptive approaches*, Van Nostrand Reinhold Company.