



A unified framework for stochastic and dynamic programming

Warren B. Powell
Princeton University

This article appeared in the Informs Computing Society Newsletter, Fall, 2012, Yongpei Guan (ed.), Copyright (c) 2012 Institute for Operations Research and the Management Sciences.

Stochastic programming and approximate dynamic programming have evolved as competing frameworks for solving sequential stochastic optimization problems, with proponents touting the strengths of their favorite approaches. With less visibility in this particular debate are communities working under names such as reinforcement learning, stochastic control, stochastic search and simulation-optimization, to name just a few. Put it all together and you get what I have come to call the jungle of stochastic optimization.

The competing communities working in stochastic optimization reflect the diversity of applications which arise in different problem settings, resulting in the development of parallel concepts, terminology and notation. Problem classes are distinguished by the nature of the decisions (discrete/continuous, scalar/vector), the underlying stochastic process, the transition function (known/unknown) and the objective function (convex? continuous?). Communities have evolved methods that are well suited to the problem classes that interest them. In the process, differences in vocabulary have hidden parallel developments (two communities doing the same thing with different terminology and vocabulary).

These differences have hidden important contributions that might help other communities. Computer scientists have ignored the power of convexity to solve problems with vector-valued actions. At the same time, the stochastic programming community has ignored the power of machine learning to approximate high-dimensional functions [8]. Years ago, I found that combining these two central ideas made it possible to solve a stochastic dynamic program with a decision vector with 50,000 dimensions and a state variable with 10^{20} dimensions [15]. In another problem, the same methods solved a stochastic, dynamic program with 175,000 time periods [11].

Stochastic programming, dynamic programming, and stochastic search can all be viewed in a unified framework if presented using common terminology and notation. One of the biggest challenges is the lack of a widely accepted modeling framework of the type that has defined the field of deterministic math programming. Misconceptions about the meaning of terms such as “state variable” and “policy” have limited dynamic programming to a relatively narrow problem class. For

this reason, I will begin with a proposal for a common modeling framework which is designed to duplicate the elegance of “ $\min cx$ subject to $Ax = b, x \geq 0$ ” that is so familiar to the operations research community. I then turn to the issue of defining what is meant by the word “policy.” This article draws heavily on the ideas in [10].

Modeling stochastic, dynamic programs There are five elements to almost any stochastic, dynamic program: States, actions, exogenous information, the transition function, and the objective function.

- The state variable S_t - Incredibly, the dynamic programming literature seems to universally use the concept of a state variable without actually defining it. In [9][Chapter 5](downloadable from <http://adp.princeton.edu>), I suggest the following definition:

Definition: 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 cost/contribution function.

It is useful to think of three types of state variables:

- The physical/resource state R_t - This captures the controllable elements of the problem which, in operations research, often refers to resources being managed. This might be the amount of inventory, the location of an aircraft, the status of a machine or the price of a stock. It might also be called the physical state, as in the location and velocity of a robot.
- The information state I_t - I_t includes exogenous information that is needed to compute the cost/contribution function, the decision function (such as the constraints), and the transition function. I_t would include exogenous information (such as market demands and wind speeds), as well as any relevant information from history that has already been observed and which is needed to make decisions in the future.
- The knowledge state K_t (also known as the belief state) - This is a set of probability distributions describing our knowledge about unobservable parameters. In some problem classes such as the multiarmed bandit problem, the belief state is the only state variable.

The stochastic programming community maintains a strict division between R_t , which always appears as a right-hand side constraint, and I_t , which is exogenous information that is not affected by current or prior decisions. However, there are applications where I_t is directly or indirectly influenced by decisions (large values of sales may influence random market prices).

An important concept is the *post-decision state* which we denote S_t^x (if our decision is x) or S_t^a (if we are using action

a). This is the state at time t immediately after a decision is made, which includes *only* the decision needed to compute the transition function (since we have already computed the cost and made a decision), which has to capture the information needed for future decisions.

By requiring that the state variable be both necessary and sufficient, all stochastic dynamic systems are Markovian by construction. At the same time, we are going to open a fresh line of thinking for the stochastic programming community that depends on scenario trees which captures the entire history.

- Decision variables - It is useful to adopt three notational systems for decision variables: a_t for discrete actions (popular in computer science and the Markov decision process community), u_t for low-dimensional, continuous controls, and x_t for discrete or continuous but typically vector-valued decisions for problems common in operations research.

When we specify a model, we avoid the problem of determining a decision other than to specify that it is the result of a *policy* which is a function that maps states to actions. We treat the determination of policies as separate from the model. While it is common to refer to a policy as $\pi(s)$, I prefer to write it as a function $X_t^\pi(S_t)$ (if it is a time-dependent function to determine x_t) or $A^\pi(s)$ (if it is a stationary policy to determine an action a). In this notation, π determines the type of function and any tunable parameters.

- Exogenous information W_t - There seems to be no common notation for the random variables in a stochastic system. Control theorists like w_t , probabilists prefer capital letters, so W_t seems like a compromise. However, unlike the control theory community, we insist that any variable indexed by t is known (“measurable”) at time t . I have also adopted the notational convention of putting hats on specific random variables, such as \hat{D}_{t+1} for new customer demands or \hat{p}_{t+1} for the change in prices.
- Transition function - Also known as the system model, state model, plant model, transfer function or just “model,” this describes the evolution of the system over time, written as

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

Note that S_t is assumed to be fully known at time t , x_t is a decision that depends on the known information in S_t , while W_{t+1} is random at time t .

In operations research, the transition function is often written as a system of linear equations, although only for the portion of the state variable that is controllable. We might write the resource transition function as

$$R_{t+1} = S^R(R_t, x_t, W_{t+1}) = A_t x_t + \hat{R}_{t+1},$$

while the information transition function $I_{t+1} = S^I(I_t, \cdot, W_{t+1})$ might represent systems of equations such as

$$\begin{aligned} p_{t+1} &= p_t + \hat{p}_{t+1}, & \text{prices} \\ E_{t+1} &= E_t + \hat{E}_{t+1}. & \text{energy from wind.} \end{aligned}$$

There are many applications in engineering (such as modeling a chemical plant) where the transition function might consist of “500 lines of Matlab code.” In others (such as modeling the effect of tax policy on climate change) the transition function is completely unknown. Problems with an unknown transition function are referred to as “model free.”

- The objective function - Using the standard notation of dynamic programming, we write the cost function as $C(S_t, x_t)$ to reflect the possible dependence of costs on the state variable (costs, such as prices, can be stochastic). Assuming we are minimizing the expected sum of costs, we would write the objective function as

$$\min_{\pi \in \Pi} \mathbb{E} \sum_{t=0}^T C(S_t, X_t^\pi(S_t)). \quad (1)$$

We have written the objective (1) assuming an undiscounted, finite horizon formulation, which is more familiar to the stochastic programming community, but we could have adopted a discounted, infinite horizon objective, or one that minimized average costs. The objection function in (1) must be accompanied by the transition function, and a model of the exogenous process for W_1, W_2, \dots

There is a tendency to equate “dynamic programming” with Bellman’s optimality equation. In fact, a number of authors will “model” a dynamic program by writing out Bellman’s optimality equation:

$$V(s) = \min_a \left(C(s, a) + \gamma \sum_{s'} P(s'|s, a) V(s') \right).$$

A dynamic program, however, is a sequential decision problem, given by (1). Bellman’s optimality equation is a way of mathematically characterizing an optimal policy and b) a strategy for designing approximate policies. The challenge we now face is the problem of specifying what it means to search over the policies $\pi \in \Pi$.

Policies The dynamic programming community is very familiar with the word “policy” which is widely understood to mean a mapping from states to actions. This is one reason why the state variable *must* include *all* the information needed to make a decision, compute the cost function and compute the transition function. There are many papers in dynamic programming where the policy is written in a lookup table form: given a discrete state s , here is the discrete action a . However,

there is no reason to limit the definition of policies to such a restricted format.

Policies come in many forms, but in my work I have been able to divide them into four fundamental classes:

- Myopic cost function approximations - A myopic policy is of the form

$$X^M(S_t) = \arg \min_{x_t \in \mathcal{X}_t} C(S_t, x_t),$$

where \mathcal{X}_t is the feasible region which may depend on S_t . In some settings, we can modify the problem to get better results over time, either by modifying the cost function itself, or possibly by modifying the constraints. We can represent this using

$$X^{CFA}(S_t|\theta) = \arg \min_{x_t \in \mathcal{X}_t(\theta)} C^\pi(S_t, x_t|\theta).$$

where θ represents any tunable parameters needed to adjust the function.

- Lookahead policies - Also known as rolling/receding horizon procedures, tree search, roll-out policies and, in control theory, model-predictive control, lookahead policies are popular in engineering practice. A deterministic lookahead policy involves solving

$$X_t^{LA-Det}(S_t) = \arg \min_{x_t \in \mathcal{X}_t} \left(c_t x_{tt} + \sum_{t'=t+1}^{t+H} c_{t'} x_{t't'} \right), \quad (2)$$

where $\arg \min_{x_t}$ optimizes over the entire (deterministic) vector $x_t = (x_{tt}, \dots, x_{t't'}, \dots, x_{t,t+H})$ over a specified horizon H , but the decision function $X_t^{LA-Det}(S_t)$ captures only x_{tt} . Of course, this has to be solved subject to constraints at each point in time and across time periods (we represent these constraints in \mathcal{X}_t). The stochastic programming community likes to incorporate uncertainty in the lookahead model, and solves

$$X_t^{LA-SP}(S_t) = \arg \min_{x_t} \left(c_t x_{tt} + \sum_{\omega \in \hat{\Omega}_t} p(\omega) \sum_{t'=t+1}^{t+H} c_{t'}(\omega) x_{t't'}(\omega) \right). \quad (3)$$

Here, $\hat{\Omega}_t$ represents a subset of random outcomes over the interval t to $t+H$. Equation (3) is a classical two-stage stochastic programming formulation, where we first choose x_{tt} , then observe ω (which might be a sequence of random variables over time periods $t+1, \dots, t+H$), and then choose $x_{t't'}(\omega)$ for all $t' > t$ given ω .

- Policy function approximations - PFAs are used when the structure of the policy $X^\pi(S_t)$ (or more likely $A^\pi(S_t)$) can be written in some analytic form. One example is

our (q, Q) inventory re-ordering policy which we can write

$$A^\pi(R_t|\theta) = \begin{cases} 0 & \text{If } R_t \geq q, \\ Q - R_t & \text{If } R_t < q, \end{cases} \quad (4)$$

where $\theta = (q, Q)$. An alternative is to use a statistical model. If x_t and S_t are scalar, we might use

$$X^\pi(S_t|\theta) = \theta_0 + \theta_1 S_t + \theta_2 S_t^2.$$

Other examples include computing a control u_t using a statistical model such as a neural network (popular in engineering). The distinguishing feature of a PFA is that it does not have an imbedded optimization problem.

- Policies based on value function approximations - VFA policies are based on Bellman's equation, and have the form

$$X_t^{VFA}(S_t) = \arg \min_{x_t \in \mathcal{X}_t} (C(S_t, x_t) + \bar{V}_t(S_t^x)). \quad (5)$$

Here, we have used the concept of the post-decision state variable S_t^x which avoids the imbedded expectation common in dynamic programming. A strategy that has attracted considerable attention under the name "approximate dynamic programming" (but studied widely in the reinforcement learning community) is to approximate $\bar{V}_t(S_t^x)$ using a linear model of the form

$$\bar{V}_t(S_t^x) = \sum_{f \in \mathcal{F}} \theta_f \phi_f(S_t^x), \quad (6)$$

where the independent variables $\phi_f(S_t^x)$ are known as "basic functions" or "features." However, a value function approximation can be represented using Benders cuts, linear approximations (linear in the state variable), or piecewise linear, separable approximations.

Three of the four classes of policies involve the use of a function approximation (cost function, policy function and value function). There are three fundamental ways of approximating functions: lookup tables (a special case), parametric models and nonparametric models. While lookup tables can be written as a parametric model, computationally they behave more like nonparametric models. In addition, a powerful class of models are known as semiparametrics, which are basically a hybrid of parametric and nonparametric. Not to be overlooked are approximation strategies aimed specifically at approximating convex problems which include strategies such as Benders' cuts, piecewise linear functions and shape-restricted statistical learning methods.

It is possible (and popular) to mix and match these fundamental strategies to create hybrids. Examples include a deterministic lookahead policy with value functions as terminal rewards, lookahead policies using cost function approximations,

value functions with policy function approximations [17], and stochastic programming with scenario trees and policy function approximations [2].

The search over policies $\pi \in \Pi$ in equation (1) might first involve a search over major policy classes (including hybrids). Then, each class of policy (or hybrid) tends to be characterized by a vector of parameters that we call θ . In a stochastic programming model, θ might include the planning horizon or the strategy for generating scenarios. In a policy function approximation, θ would be the parameters (or rules) that specify the policy.

It is possible that someone has worked with a policy that does not fit any of these four (or hybrids), but this is what I have encountered in my own work. It is easy to describe problems that are particularly well-suited to each of these four classes. The important idea is to get away from equating dynamic programming with lookup tables.

With our new vocabulary and perspective, it is possible to identify close relationships between communities that have previously been viewed as completely distinct. Below I show how to link stochastic search and dynamic programming, and then dynamic programming and stochastic programming.

From stochastic search to dynamic programming

Problems in stochastic search are typically written

$$\min_x \mathbb{E}F(x, W), \quad (7)$$

where x is a deterministic set of parameters and W is a random variable. Stochastic search has been viewed as distinctly different from sequential decision problems (dynamic programs) because decisions x_t in sequential problems are random variables in the future. However, this is not the right way to look at them. In a dynamic program, the optimization problem is (in most cases) a search for a deterministic *policy*. We might write

$$F^\pi(S_0) = \mathbb{E}\hat{F}^\pi(S_0, W) = \sum_{t=0}^T C(S_t, X_t^\pi(S_t)),$$

where $S_{t+1} = S^M(S_t, X_t^\pi(S_t), W_{t+1})$. The optimization problem is then

$$\min_\pi \mathbb{E}\hat{F}^\pi(S_0, W). \quad (8)$$

Finding the best deterministic policy (which includes finding both the structure of the policy and any parameters that characterize the policy) corresponds directly to the stochastic search problem in (7).

This perspective opens up a powerful set of tools for solving dynamic programs. For example, imagine that we have a policy given by

$$X_t^{VFA}(S_t|\theta) = \arg \min_{x_t \in \mathcal{X}_t} \left(C(S_t, x_t) + \sum_{f \in \mathcal{F}} \theta_f \phi_f(S_t^x) \right).$$

While there is a substantial literature that tries to estimate θ using Bellman error minimization (so that $\bar{V}(S_t)$ predicts the value of being in a state), growing attention has been given to the idea of directly searching for the best value of θ to minimize costs. This is the same as solving equation (8) with $\pi = \theta$ which, of course, is the same as solving equation (7).

Stochastic search (known as *direct policy search* in the ADP/RL literature), can work extremely well, but it is not well suited to all problems. If we are unable to compute derivatives with respect to θ , then algorithms for derivative-free stochastic optimization generally limit the number of dimensions of θ . Particularly difficult are time-dependent problems where θ_t is a function of time. However, anyone working in dynamic programming should have a working knowledge of the tools of stochastic search.

From dynamic programming to stochastic programming

The transition from dynamic programming to stochastic programming is somewhat more difficult if done carefully. There has been growing recognition of the links between stochastic programming and dynamic programming (see [3] and [13]). Thorough presentations of stochastic programming can be found in [5], [1], and [14].

We start with the classical statement of Bellman's equation for discrete state and action spaces (see [12])

$$V(s) = \min_{a \in \mathcal{A}} \left(C(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V(s') \right).$$

The stochastic programming community works purely with time-dependent, finite horizon problems which we can write as

$$V_t(S_t) = \min_{a_t \in \mathcal{A}_t} \left(C(S_t, a_t) + \gamma \sum_{s' \in \mathcal{S}} P(S_{t+1} = s' | S_t, a_t) V_{t+1}(s') \right).$$

We next replace the one-step transition matrix with an expectation, giving us

$$V_t(S_t) = \min_{a_t \in \mathcal{A}_t} (C(S_t, a_t) + \gamma \mathbb{E}\{V_{t+1}(S_{t+1}) | S_t\}), \quad (9)$$

where $S_{t+1} = S^M(S_t, a_t, W_{t+1})$. There is an implicit assumption that we are using an optimal policy starting at time $t + 1$ onward. We are going to temporarily fix the policy represented by $A_{t'}^\pi(S_{t'})$, and replace the value function with the expected sum of costs from $t + 1$ and the end of the planning horizon,

$$V_t^\pi(S_t) = \min_{a_t \in \mathcal{A}_t} \left(C(S_t, a_t) + \mathbb{E} \left\{ \sum_{t'=t+1}^T \gamma^{t'-t} C(S_{t'}, A_{t'}^\pi(S_{t'})) \mid S_t \right\} \right).$$

We cannot compute the expectation, so we are going to replace it with a Monte Carlo sample. This is also a good time to

switch from our discrete action a_t to vector-valued actions x_t . This gives us

$$\bar{V}_t^\pi(S_t) = \min_{x_t \in \mathcal{X}_t} \left(C(S_t, x_t) + \frac{1}{|\hat{\Omega}|} \sum_{\omega \in \hat{\Omega}} \sum_{t'=t+1}^T \gamma^{t'-t} C(S_{t'}(\omega), X_{t'}^\pi(S_{t'}(\omega))) \right).$$

The state $S_{t'}$ consists of two components: the resource state $R_{t'}$ that is controlled by our decisions $x_t, \dots, x_{t'}$, and the exogenous information that we previously introduced as $I_{t'}$. In stochastic programming, it is common practice to view the exogenous information state as the entire history $h_{t'}$, and we will continue to reference the history $h_{t'}$ to develop the link with stochastic programming. Thus, we can write $S_{t'} = (R_{t'}, I_{t'}) = (R_{t'}, h_{t'})$.

We are primarily interested in making a decision at time t , so we are going to make the transition from approximating a policy over the entire simulation to time T to solving a *lookahead model* over the interval t to $t+H$, where H is the planning horizon. For example, we may want to model an energy storage problem over an entire year by solving sequences of 24-hour optimization problems. However, even this shorter horizon problem can be exceptionally difficult, so we have to develop special algorithmic strategies.

Remembering that we are starting at time t in a given initial state S_t , we can write our history as

$$h_{t'} = (S_t, W_{t+1}, W_{t+2}, \dots, W_{t'}),$$

where t' ranges from t to $t+H$. We are going to drop the reference to the unspecified policy $X_{t'}^\pi(S_{t'})$ and allow ourselves to choose actions in the future optimally, given the histories $h_{t'}(\omega)$, $\omega \in \hat{\Omega}$. At this point we are also going to recognize that this optimization problem is a lookahead policy determined by solving the optimization problem

$$X_t^\pi(S_t) = \arg \min_{x_t} \left(C(S_t, x_t) + \min_{x_{t+1}, \dots, x_{t+H}} \frac{1}{|\hat{\Omega}|} \sum_{\omega \in \hat{\Omega}} \sum_{t'=t+1}^{t+H} \gamma^{t'-t} C(S_{t'}(\omega), x_{t'}(h_{t'}(\omega))) \right),$$

where there is a vector $x_{t'}$ for each history $h_{t'}(\omega)$, and where the decision x_t affects the constraints for x_{t+1}, \dots, x_{t+H} . We are now solving a stochastic optimization problem over a limited horizon H and a limited set of outcomes $\hat{\Omega}$, and yet even this problem is quite difficult. We are going to make one last tweak to get it into the more compact form

$$X_t^\pi(S_t) = \arg \min_{x_t, \dots, x_{t+H}} \frac{1}{|\hat{\Omega}|} \sum_{\omega \in \hat{\Omega}} \sum_{t'=t}^{t+H} \gamma^{t'-t} C(S_{t'}(\omega), x_{t'}(\omega)), \quad (10)$$

where $x_t, \dots, x_{t+H} = (x_t(\omega), \dots, x_{t+H}(\omega))$, $\forall \omega \in \hat{\Omega}$. We need to solve this optimization problem subject to the con-

straints for $t' = t+1, \dots, t+H$ and all $\omega \in \hat{\Omega}$,

$$A_t x_t(\omega) = b_t, \quad (11)$$

$$x_t(\omega) \geq 0, \quad (12)$$

$$A_{t'}(\omega) x_{t'}(\omega) - B_{t'-1}(\omega) x_{t'-1}(\omega) = b_{t'}(\omega), \quad (13)$$

$$x_{t'}(\omega) \geq 0. \quad (14)$$

In this formulation, we interpret $x_{t'}(\omega)$ exactly as it is written - a single vector $x_{t'}$ for each outcome $\omega \in \hat{\Omega}$ rather than the history $h_{t'}(\omega)$. Instead of having one vector $x_{t'}$ for each history $h_{t'}$ (which is a node in the scenario tree), we now have a vector $x_{t'}$ for each $\omega \in \hat{\Omega}$. This creates a problem that we are allowed to see into the future, so we add the *nonanticipativity constraints*

$$x_{t'}(\omega) - \bar{x}_{t'}(h_{t'}) = 0, \quad \forall \omega \in \mathcal{H}_{t'}(h_{t'}), \quad (15)$$

where $\mathcal{H}_{t'}(h_{t'})$ is the set of outcomes $\omega \in \hat{\Omega}$ where the observations $W_{t+1}, \dots, W_{t'}$ match $h_{t'}$.

The challenge with this strategy is that the scenario tree (the set of histories) explodes very quickly. A common strategy is to generate a set of outcomes for W_{t+1} , and then sharply limit further branching in the scenario tree for later time periods ([1], [14], [7]). We can do this because we are only interested in the decision x_t that we are going to implement now.

Given the complexity of solving the lookahead policy, there is by now a substantial research literature that has grown up around various strategies for finding near-optimal solutions. However, it is important to realize that an optimal solution of (10) does not mean the resulting policy is optimal. Indeed, bounds on the quality of the solution to (10) do not directly translate to bounds on the performance of the policy in equation (1). This is true even if we choose $t+H = T$, because the lookahead model is limited by the use of a scenario tree.

While explicit lookahead policies are popular in stochastic programming, considerable attention has been given to using a particular class of value function approximations for solving the lookahead model. We use the notational system of [14] to work backward from a formulation used in stochastic programming to the notation we have been using in approximate dynamic programming. This progression starts with

$$Q_t(x_{t-1}, \xi_{[t]}) = \min_{x_t} (c_t x_t + \mathbb{E}\{Q_{t+1}(x_t, \xi_{[t+1]}) | \xi_{[t]}\}). \quad (16)$$

Here, Q_t is called the *recourse function*, but this is just different terminology and notation for our value function V_t . $\xi_{[t]}$ is the history of the exogenous information process up to time t (which we refer to as h_t). The resource vector R_t is a function of x_{t-1} and, if we have an exogenous component such as \hat{R}_t , then it also depends on W_t (which is contained in $\xi_{[t]}$). This means that the state variable is given by $S_t = (R_t, h_t) = (x_{t-1}, \xi_{[t]})$.

We note that it is mathematically equivalent to use x_{t-1} instead of R_t , but in most applications R_t is lower dimensional

than x_{t-1} and would be more effective computationally as a state variable. Indeed, we would argue that while x_{t-1} is a sufficient statistic to describe the resource state R_t , it is not necessary because it carries more dimensions than are necessary.

These observations allow us to write (16) as

$$Q_t(x_{t-1}, \xi_{[t]}) = \min_{x_t} (c_t x_t + Q_t^x(R_t^x, h_t)) \quad (17)$$

$$= \min_{x_t} (c_t x_t + Q_t^x(x_t, h_t)). \quad (18)$$

In the notation of dynamic programming (but retaining the linear cost structure), this would be written

$$V_t(S_t) = \min_{x_t} (c_t x_t + V_t^x(S_t^x)). \quad (19)$$

Equation (18) is a deterministic optimization problem, which is much more amenable to solution using the tools of math programming. Our only challenge, then, is finding an approximation of $V_t^x(S_t^x) = Q_t^x(x_t, h_t)$. The most popular strategy in stochastic programming is to use Benders' decomposition, where $Q_t^x(x_t, h_t)$ is replaced with a series of cuts, producing the linear program

$$V_t(S_t) = \min_{x_t, v} (c_t x_t + v), \quad (20)$$

where

$$v \geq \alpha_t^k(h_t) + \beta_t^k(h_t)x_t(h_t), \quad \text{for } k = 1, \dots, K. \quad (21)$$

It is standard notation in the stochastic programming community to index these parameters as $(\alpha_{t+1}^k(h_t), \beta_{t+1}^k(h_t))$ because the cuts are approximating the problem at time $t+1$, but the parameters are \mathcal{F}_t -measurable, and for this reason it is more consistent with our notation to index them by time t .

The optimization problem (20) with (21) is a linear program indexed by the history h_t . The parameters $(\alpha_t^k(h_t), \beta_t^k(h_t))$ are generated by simulating our way to h_{t+1} from h_t , solving the optimization problem at node h_{t+1} , and then using the dual information to update the parameters $(\alpha_t^k(h_t), \beta_t^k(h_t))$. This type of updating is completely familiar to the approximate dynamic programming community. Indeed, it should be easy to see that we are approximating the recourse function around the post-decision state at time t given by $\mathbb{E}\{Q_{t+1}(x_t, \xi_{[t+1]}) | \xi_{[t]}\} = V_t^x(S_t^x) = V_t^x(R_t^x, h_t) = V_t^x(x_{t-1}, h_t)$.

While Benders' cuts are very popular in the stochastic programming community, they are hardly the only way to approximate the value of the future. An alternative strategy is to use a function that is linear in the (post-decision) resource variable, as in

$$\bar{V}_t(S_t) = \min_{x_t \in \mathcal{X}_t} \left(c_t x_t + \sum_i \bar{v}_{it} R_{it}^x \right), \quad (22)$$

Yet another strategy is to use an approximation that is piecewise linear but separable in R_t^x , as in

$$V_t(S_t) = \min_{x_t \in \mathcal{X}_t} \left(c_t x_t + \sum_i \bar{V}_{it}(R_{it}^x) \right). \quad (23)$$

Note that the approximations \bar{v}_{it} and $\bar{V}_{it}(R_{it}^x)$ are not indexed by the history h_t , making these methods computationally much more compact.

These approximation strategies highlight a critical difference that separates approximate dynamic programming from stochastic programming. The value function approximations (22) and (23) are calculated over the entire simulation $t = 0, \dots, T$ rather than just within a planning horizon $t, \dots, t+H$ (see [15] and [16] for illustrations). This can be an important property when we are interested in the value functions themselves. In [11], we estimate value function approximations of the form given in (23) for 175,000 time periods. This produces a policy of the form

$$X_t^x(S_t) = \arg \min_{x_t \in \mathcal{X}_t} \left(c_t x_t + \sum_i \bar{V}_{it}(R_{it}^x) \right), \quad (24)$$

where x_t is a vector with hundreds or even thousands of dimensions.

By contrast, the stochastic programming community is using Benders' cuts to form a "policy" for solving the lookahead model over time periods $t, \dots, t+H$. The policy that is then used for the full model is the decision produced at time t from solving the lookahead model. After this decision, we roll forward to $t+1$ and solve a new problem over the horizon $t+1$ to $t+1+H$ and then repeat the entire process.

The only reason this is necessary is because of the use of the scenario tree which is generated from the current state S_t , and which limits the lookahead model to relatively shorter horizons. If we dropped the indexing of the cuts in (21) on the nodes in the scenario tree, then it would be possible to use Benders' cuts over the entire horizon $t = 0, \dots, T$ in one large model. An alternative strategy would be to index the cuts based on clusters formed around the information state (rather than the entire history). The important idea is to draw on the tools of machine learning [4] rather than depending on lookup tables (nodes in the scenario tree).

Closing thoughts While both the stochastic programming and dynamic programming communities work on sequential decision problems, there are some important differences in cultures between the two communities. While it is important to understand differences in terminology and notation, it is also useful to understand the differences in motivating applications and research styles.

Research in the stochastic programming community seems to be characterized by:

- Works exclusively on time-dependent problems.
- Primary tools are explicit lookahead policies and value functions based on Benders' cuts to exploit convexity.
- Emphasizes convergence proofs and bounds on policies for solving an approximate lookahead model rather than the value of a policy over long horizons.

- Exploits convexity to handle vector-valued decisions.
- Depends heavily on the concept of scenario trees and lookup tables rather than general purpose statistical learning algorithms.

By contrast, the dynamic programming community seems to be characterized by:

- Most of the research seems to be on infinite horizon problems, but applications to time-dependent problems are common.
- Most attention is on policies based on value function approximations and policy function approximations.
- Focus is on finding the best policy in terms of its performance over a long (or infinite) horizon.
- More emphasis on discrete action spaces, or low-dimensional continuous controls, without assuming convexity.
- Extensive use of machine learning techniques to approximate value functions.

These are generalizations, of course, but represent my sense of these communities. My hope is that this discussion will help to bridge the gap.

References

- [1] Birge, J. R. & Louveaux, F. (1997), *Introduction to Stochastic Programming*, Springer Verlag, New York.
- [2] Defourny, B., Ernst, D. & Wehenkel, L. (2013), ‘Scenario Trees and Policy Selection for Multistage Stochastic Programming using Machine Learning’, *Inform. J. on Computing* (to appear).
- [3] Dupacova, J. & Sladky, K. (2001), ‘Comparison of Multistage Stochastic Programs with Recourse and Stochastic Dynamic Programs with Discrete Time’, *Z. Angew. Math. Mech.* **81**, 1–15.
- [4] Hastie, T., Tibshirani, R. & Friedman, J. (2009), *The elements of statistical learning: data mining, inference and prediction*, Springer, New York.
- [5] Higle, J. & Sen, S. (1996), *Stochastic Decomposition: A Statistical Method for Large Scale Stochastic Linear Programming*, Kluwer Academic Publishers.
- [6] Jacobs, J., Freeman, G., Grygier, J., Morton, D., Schultz, G., Staschus, K. & Stedinger, J. (1995), ‘SOCRATES-A system for scheduling hydroelectric generation under uncertainty’, *Ann. Oper. Res.*, **59**, 99–133.
- [7] King, A. J. & Wallace, S. (2012), *Modeling with stochastic programming*, Springer Verlag, New York.
- [8] Powell, W. B. (2009), ‘Feature Article—Merging AI and OR to Solve High-Dimensional Stochastic Optimization Problems Using Approximate Dynamic Programming’, *Inform. J. on Computing*, **22**, 2–17.
- [9] Powell, W. B. (2011), *Approximate Dynamic Programming: Solving the curses of dimensionality*, 2nd. edn, John Wiley & Sons, Hoboken, NJ.
- [10] Powell, W. B., Simao, H. P., Bouzaiene-Ayari, B. (2012), ‘Approximate Dynamic Programming in Transportation and Logistics : A Unified Framework’, *European J. of Transportation and Logistics*, (to appear).
- [11] Powell, W. B., George, A., Lamont, A. & Stewart, J. (2011), ‘SMART: A Stochastic Multiscale Model for the Analysis of Energy Resources, Technology and Policy’, *Inform. J. on Computing*.
- [12] Puterman, M. L. (1994), *Markov Decision Processes*, 1st edn, John Wiley and Sons, Hoboken.
- [13] Sen, S. & Zhou, Z. (2012), Multi-stage Stochastic Decomposition: A Sequential Sampling Algorithm for Multi-stage Stochastic Linear Programs.
- [14] Shapiro, A., Dentcheva, D. & Ruszczyński, A. (2009), *Lectures on stochastic programming: modeling and theory*, SIAM, Philadelphia.
- [15] Simao, H. P., Day, J., George, A., Gifford, T., Powell, W. B. & Nienow, J. (2009), ‘An Approximate Dynamic Programming Algorithm for Large-Scale Fleet Management: A Case Application’, *Transportation Science* **43**(2), 178–197.
- [16] Topaloglu, H. & Powell, W. B. (2006), ‘Dynamic Programming Approximations for Stochastic, Time-Staged Integer Multi-commodity Flow Problems’, *Inform. Journal on Computing* **18**, 31–42.
- [17] Wu, T., Powell, W. B. & Whisman, A. (2009), ‘The Optimizing-Simulator: An Illustration using the Military Airlift Problem’, *ACM Transactions on Modeling and Simulation* **19**(3), 1–31.