From Reinforcement Learning to Optimal Control: A unified framework for sequential decisions

Warren B. Powell
Department of Operations Research and Financial Engineering
Princeton University

December 10, 2019

Abstract

There are over 15 distinct communities that work in the general area of sequential decisions and information, often referred to as decisions under uncertainty or stochastic optimization. We focus on two of the most important fields: stochastic optimal control, with its roots in deterministic optimal control, and reinforcement learning, with its roots in Markov decision processes. Building on prior work, we describe a unified framework that covers all 15 different communities, and note the strong parallels with the modeling framework of stochastic optimal control. By contrast, we make the case that the modeling framework of reinforcement learning, inherited from discrete Markov decision processes, is quite limited. Our framework (and that of stochastic control) is based on the core problem of optimizing over policies. We describe four classes of policies that we claim are universal, and show that each of these two fields have, in their own way, evolved to include examples of each of these four classes.

1 Introduction

There is a vast range of problems that consist of the sequence: decisions, information, decisions, information, ... Application areas span engineering, business, economics, finance, health, transportation, and energy. It encompasses active learning problems that arise in the experimental sciences, medical decision making, e-commerce, and sports. It also includes iterative algorithms for stochastic search, as well as two-agent games and multiagent systems. In fact, we might claim that virtually any human enterprise will include instances of sequential decision problems.

Given the diversity of problem domains, it should not be a surprise that a number of communities have emerged to address the problem of making decisions over time to optimize some metric. The reason that so many communities exist is a testament to the variety of problems, but it also hints at the many methods that are needed to solve these problems. As of this writing, there is not a single method that has emerged to solve all problems. In fact, it is fair to say that all the methods that have been proposed are *fragile*: relatively modest changes can invalidate a theoretical result, or increase run times by orders of magnitude.

In Powell (2019), we present a unified framework for all sequential decision problems. This framework consists of a mathematical model (that draws heavily from the framework used widely in stochastic control), which requires optimizing over *policies* which are functions for making decisions given what we know at a point in time (captured by the state variable).

The significant advance of the unified framework is the identification of four (meta)classes of policies that encompass all the communities. In fact, whereas the solution approach offered by each community is fragile, we claim that the four classes are universal: any policy proposed for any sequential decision problem will consist of one of these four classes, and possibly a hybrid.

The contribution of the framework is to raise the visibility of all of the communities. Instead of focusing on a specific solution approach (for example, the use of Hamilton-Jacobi-Bellman (HJB) equations, which is one of the four classes), the framework encourages people to consider all four classes, and then to design policies that are best suited to the characteristics of a problem.

This chapter is going to focus attention on two specific communities: stochastic optimal control, and reinforcement learning. Stochastic optimal control emerged in the 1950's, building on what was already a mature community for deterministic optimal control that emerged in the early 1900's and has been adopted around the world. Reinforcement learning, on the other hand, emerged in the 1990's building on the foundation of Markov decision processes which was introduced in the 1950's (in fact, the first use of the term "stochastic optimal control" is attributed to Bellman, who invented Markov decision processes). Reinforcement learning emerged from computer science in the 1980's, and grew to prominence in 2016 when it was credited with solving the Chinese game of Go using AlphaGo.

We are going to make the following points:

- Both communities have evolved from a core theoretical/algorithm result based on Hamilton-Jacobi-Bellman equations, transitioning from exact results (that were quite limited), to the use of algorithms based on approximating value functions/cost-to-go functions/Q-factors, to other strategies that do not depend on HJB equations. We will argue that each of the fields is in the process of recognizing all four classes of policies.

- We will present and contrast the canonical modeling frameworks for stochastic control and reinforcement learning (adopted from Markov decision processes). We will show that the framework for stochastic control is very flexible and scalable to real applications, while that used by reinforcement learning is limited to a small problem class.

- We will present a universal modeling framework for sequential decision analytics (given in Powell (2019)) that covers any sequential decision problem. The framework draws heavily from that used by stochastic control, with some minor adjustments. While not used by the reinforcement learning community, we will argue that it is used implicitly. In the process, we will dramatically expand the range of problems that can be viewed as either stochastic control problems, or reinforcement learning problems.

We begin our presentation in section 2 with an overview of the different communities that work on sequential decisions under uncertainty, along with a list of major problem classes. Section 3 presents a side-by-side comparison of the modeling frameworks of stochastic optimal control and reinforcement learning.

Section 4 next presents our universal framework (taken from Powell (2019)), and argues that a) it covers all 15+ fields (presented in section 2) dealing with sequential decisions and uncertainty, b) it draws heavily from the standard model of stochastic optimal control, and c) the framework of reinforcement learning, inherited from discrete Markov decision processes, has fundamental weaknesses that limit its applicability to a very narrow classes of problems. We then illustrate the framework using an energy storage problem in section 5; this application offers tremendous richness, and allows us to illustrate the flexibility of the framework.

The central challenge of our modeling framework involves optimizing over policies, which represents our point of departure with the rest of the literature, since it is standard to pick a class of policy in advance. However, this leaves open the problem of how to search over policies. In section 6 we present four (meta)classes of policies which, we claim, are universal, in that *any* approach suggested in the literature (or in practice) is drawn from one of these four classes, or a hybrid of two or more. Section 7 illustrates all four classes, along with a hybrid, using the context of our energy storage application. These examples will include hybrid resource allocation/active learning problems, along with the overlooked challenge of dealing with rolling forecasts.

Section 8 briefly discusses how to use the framework to model multiagent systems, and notes that this vocabulary provides a fresh perspective on partially observable Markov decision processes. Then, section 9 concludes the chapter with a series of observations about reinforcement learning, stochastic optimal control, and our universal framework.

2 The communities of sequential decisions

The list of potential applications of sequential decision problems is virtually limitless. Below we list a number of major application domains. Ultimately we are going to model all of these using the same framework.

Discrete problems These are problems with discrete states and discrete decisions (actions), such as stochastic shortest path problems.

Control problems These span controlling robots, drones, rockets and submersibles, where states are continuous (location and velocity) as are controls (forces). Other examples include determining optimal dosages of medications, or continuous inventory (or storage) problems that arise in finance and energy.

Dynamic resource allocation problems Here we are typically managing inventories (retail products, food, blood, energy, money, drugs), typically over space and time. It also covers discrete problems such as dynamically routing vehicles, or managing people or machines. It would also cover planning the movements of robots and drones (but not how to do it). The scope of "dynamic resource allocation problems" is almost limitless.

Active learning problems This includes any problem that involves learning, and where decisions affect what information is collected (laboratory experiments, field experiments, test marketing, computer simulations, medical testing). It spans multiarmed bandit problems, e-commerce (bidding, recommender systems), black-box simulations, and simulation-optimization.

Hybrid learning/resource allocation problems This would arise if we are managing a drone that is collecting information, which means we have to manage a physical resource while running experiments which are then used to update beliefs. Other problems are laboratory science experiments with setups (this is the physical resource), collecting public health information from field technicians, and any experimental learning setting with a budget constraint.

Stochastic search This includes both derivative-based and derivative-free stochastic optimization.

Adversarial games This includes any two-player (or multiplayer) adversarial games. It includes pricing in markets where price affects market behavior, and military applications.

Multiagent problems This covers problems with multiple decision-makers who might be competing or cooperating. They might be making the same decisions (but spatially distributed), or making different decisions that interact (as arises in supply chains, or where different agents play different roles but have to work together).

Given the diversity of problems, it should not be surprising that a number of different research communities have evolved to address them, each with their own vocabulary and solution methods, creating what we have called the "jungle of stochastic optimization" (Powell (2014), see also `jungle.princeton.edu`). A list of the different communities that address the problem of solving sequential decision-information problems might be:

- Stochastic search (derivative-based)
- Ranking and selection (derivative-free)
- (Stochastic) optimal control
- Markov decision processes/dynamic programming
- Simulation-optimization
- Optimal stopping
- Model predictive control
- Stochastic programming
- Chance-constrained programming
- Approximate/adaptive/neuro-dynamic programming
- Reinforcement learning
- Robust optimization
- Online computation
- Multiarmed bandits
- Active learning
- Partially observable Markov decision processes

Each of these communities is supported by at least one book and over a thousand papers.

Some of these fields include problem classes that can be described as static: make decision, see information (possibly make one more decision), and then the problem stops (stochastic programming and robust optimization are obvious examples). However, all of them include problems that are fully sequential, consisting of sequences of decision, information, decision, information, ..., over a finite or infinite horizon. The focus of this chapter is on fully sequential problems.

Several of the communities offer elegant theoretical frameworks that lead to optimal solutions for specialized problems (Markov decision processes and optimal control are two prominent examples).

Others offer asymptotically optimal algorithms: derivative-based and certain derivative-free stochastic optimization problems, simulation-optimization, and certain instances of approximate dynamic programming and reinforcement learning. Still others offer theoretical guarantees, often in the form of regret bounds (that is, bounds on how far the solution is from optimal).

We now turn our attention to focus on the fields of stochastic optimal control and reinforcement learning.

3 Stochastic optimal control vs. reinforcement learning

There are numerous communities that have contributed to the broad area of modeling and solving sequential decision problems, but there are two that stand out: optimal control (which laid the foundation for stochastic optimal control), and Markov decision processes, which provided the analytical foundation for reinforcement learning. Although these fields have intersected at different times in their history, today they offer contrasting frameworks which, nonetheless, are steadily converging to common solution strategies.

We present the modeling frameworks of (stochastic) optimal control and reinforcement learning (drawn from Markov decision processes), which are polar opposites. Given the growing popularity of reinforcement learning, we think it is worthwhile to compare and contrast these frameworks. We then present our own universal framework which spans all the fields that deal with any form of sequential decision problems. The reader will quickly see that our framework is quite close to that used by the (stochastic) optimal control community, with a few adjustments.

3.1 Stochastic control

The field of optimal control enjoys a long and rich history, as evidenced by the number of popular books that have been written focusing on deterministic control, including Vrabie & Lewis (2009), Kirk (2004), and Stengel (1994). There are also a number of books on stochastic control (see Sethi (2019), Nisio (2014), Sontag (1998), Stengel (1986), Bertsekas & Shreve (1978), Kushner (1971)) but these tend to be mathematically more advanced.

Deterministic optimal control problems are typically written

$$\min_{u_0,\ldots,u_T} \sum_{t=0}^{T-1} L_t(x_t, u_t) + L_T(x_T), \tag{1}$$

where x_t is the state at time t, u_t is the control (that is, the decision) and $L_t(x_t, u_t)$ is a loss function

with terminal loss $L_T(x_T)$. The state x_t evolves according to

$$x_{t+1} = f_t(x_t, u_t), \tag{2}$$

where $f_t(x_t, u_t)$ is variously known as the transition function, system model, plant model (as in chemical or power plant), plant equation, and transition law. We write the control problem in discrete time, but there is an extensive literature where this is written in continuous time, and the transition function is written

$$\dot{x}_t = f_t(x_t, u_t).$$

The most common form of a stochastic control problem simply introduces additive noise to the transition function given by

$$x_{t+1} = f_t(x_t, u_t) + w_t, \tag{3}$$

where w_t is random at time t. This odd notation arose because of the continuous time formulation, where w_t would be disturbances (such as wind pushing against an aircraft) between t and $t + dt$. The introduction of the uncertainty means that the state variable x_t is a random variable when we are sitting at time 0. Since the control u_t is also a function of the state, this means that u_t is also a random variable. Common practice is to then take an expectation of the objective function in equation (1), which produces

$$\min_{u_0, \dots, u_T} \mathbb{E} \left\{ \sum_{t=0}^{T-1} L_t(x_t, u_t) + L_T(x_T) \right\}, \tag{4}$$

which has to be solved subject to the constraint in equation (3). This is problematic, because we have to interpret (3) recognizing that x_t and u_t are random since they depend on the sequence $w_0, \dots, w_{t-1}$.

In the deterministic formulation of the problem, we are looking for an optimal control vector $u_0^*, \dots, u_T^*$. When we introduce the random variable w_t, then the controls need to be interpreted as functions that depend on the information available at time t. Mathematicians handle this by saying that "u_t must be $\mathcal{F}_t$-measurable" which means, in plain English, that the control u_t is a function (not a variable) which can only depend on information up through time t. This leaves us the challenge of finding this function.

We start by relaxing the constraint in (3) and add it to the objective function, giving us

$$\min_{u_0, \dots, u_T} \mathbb{E} \left\{ \sum_{t=0}^{T-1} L_t(x_t, u_t) + L_T(x_T) + \lambda_t(f(x_t, u_t) + w_t - x_{t+1}) \right\}. \tag{5}$$

where $(\lambda_t)_{t=0}^T$ is a vector of dual variables (known as costate variables in the controls community). Assuming that $\mathbb{E}w_t = 0$, then w_t drops out of the objective function.

The next step is that we restrict our attention to quadratic loss functions given by

$$L_t(x_t, u_t) = (x_t)^T Q_t x_t + (u_t)^T R_t u_t, \tag{6}$$

where Q_t and R_t are a set of known matrices. This special case is known as linear-quadratic regulation (or LQR).

With this special structure, we turn to the Hamilton-Jacobi equations (often called the Hamilton-Jacobi-Bellman equations) where we solve for the "cost-to-go" function $J_t(x_t)$ using

$$J_t(x_t) = \min_u \left(L_t(x_t, u) + \mathbb{E}_w J_{t+1}(f(x_t, u, w)) \right). \tag{7}$$

$J_t(x_t)$ is the value of being in state x_t at time t and following an optimal policy from time t onward. In the language of reinforcement learning, $J_t(x_t)$ is known as the value function, and is written $V_t(S_t)$.

For the special case of the quadratic objective function in (6), it is possible to solve the Hamilton-Jacobi-Bellman equations analytically and show that the optimal control as a function of the state is given by

$$u_t = K_t x_t, \tag{8}$$

where K_t is a matrix that depends on Q_t and R_t.

Here, u_t is a *function* that we refer to as a *policy* π, but is known as a *control law* in the controls community. Some would write (8) as $\pi_t(x_t) = K_t x_t$. Later we are going to adopt the notation $U^\pi(x_t)$ for writing a policy, where π carries information about the structure of the policy. We note that when the policy depends on the state x_t, then the function is, by construction, "$\mathcal{F}_t$-measurable," so we can avoid this terminology entirely.

The linear control law (policy) in equation (8) is very elegant, but it is a byproduct of the special structure, which includes the quadratic form of the objective function (equation (6)), the additive noise (equation (3)), and the fact that there are no constraints on the controls. For example, a much more general way of writing the transition function is

$$x_{t+1} = f(x_t, u_t, w_t),$$

which allows the noise to enter the dynamics in any form. For example, consider an inventory problem where the state (the inventory level) is governed by

$$x_{t+1} = \max\{0, x_t + u_t - w_{t+1}\},$$

where w_{t+1} is the random demand for our product.

We are also interested in general state-dependent reward functions which are often written as $g(x_t, u_t)$ (where $g(\cdot)$ stands for gain), as well as the constraints, where we might write

$$
\begin{aligned}
A_t u_t &= b_t, \\
u_t &\geq 0,
\end{aligned}
$$

where b_t (and A_t) may contain information from the state variable.

For these more general problems, we cannot compute (7) exactly, so the research community has developed a variety of methods for approximating $J_t(x_t)$. Methods for solving (7) approximately have been widely studied in the controls community under names such as heuristic dynamic programming, approximate dynamic programming, neuro-dynamic programming, and adaptive dynamic programming. However, even this approach is limited to special classes of problems within our universe of sequential decision problems.

Optimal control enjoys a rich history. Deterministic control dates to the early 1900's, while stochastic control appears to have been first introduced by Bellman in the 1950's (known as the father of dynamic programming). Some of the more recent books in optimal control are Kirk (2004), Stengel (1986), Sontag (1998), Sethi (2019), and Lewis et al. (2012). The most common optimal control problems are continuous, low-dimensional and unconstrained. Stochastic problems are most typically formulated with additive noise.

The field of stochastic control has tended to evolve using the more sophisticated mathematics that has characterized the field. Some of the most prominent books include Astrom (1970), Kushner (1971), Bertsekas & Shreve (1978), Yong & Zhou (1999), Nisio (2014) (note that some of the books on deterministic controls touch on the stochastic case).

We are going to see below that this framework for writing sequential decision problems is quite powerful, even if the classical results (such as the linear control policy) are very limited. It will form the foundation for our unified framework, with some slight adjustments.

3.2 Reinforcement learning

The field known as reinforcement learning evolved from early work done by Rich Sutton and his adviser Andy Barto in the early 1980's. They addressed the problem of modeling the search process of a mouse exploring a maze, developing methods that would eventually help solve the Chinese game of Go, outperforming world masters (figure 1).

Sutton and Barto eventually made the link to the field of Markov decision processes and adopted the vocabulary and notation of this field. The field is nicely summarized in Puterman (2005) which
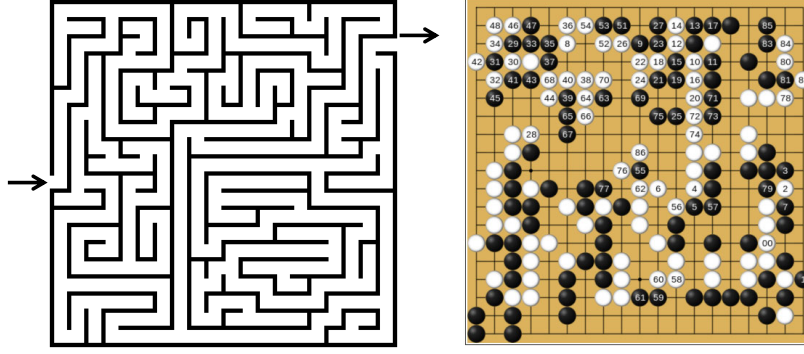
Figure 1: From the mouse-in-the-maze problem, to Chinese Go - a trajectory of reinforcement learning.

can be viewed as the capstone volume on 50 years of research into Markov decision processes, starting with the seminal work of Bellman (Bellman 1957). Puterman (2005)[Chapter 3] summarizes the modeling framework as consisting of the following elements:

Decision epochs $T = 1, 2, \ldots, N$.

State space $\mathcal{S}$ = set of (discrete) states.

Action space $\mathcal{A}$ = action space (set of actions when we are in state s).

Transition matrix $p(s'|s, a)$ = probability of transitioning to state s' given that we are in state s and take action a.

Reward $r(s, a)$ = the reward received when we are in state s and take action a.

This notation (which we refer to below as the "MDP formal model") became widely adopted in the computer science community where reinforcement learning evolved. It became standard for authors to define a reinforcement learning problem as consisting of the tuple $(\mathcal{S}, \mathcal{A}, P, r)$ where P is the transition matrix, and r is the reward function.

Using this notation, Sutton and Barto (this work is best summarized in their original volume Sutton & Barto (1998)) proposed estimating the value of being in a state s^n and taking an action a^n (at the nth iteration of the algorithm) using

$$\hat{q}^n(s^n, a^n) = r(s^n, a^n) + \gamma \max_{a' \in \mathcal{A}_{s'}} \bar{Q}^{n-1}(s', a'), \tag{9}$$

$$\bar{Q}^n(s^n, a^n) = (1 - \alpha_n)\bar{Q}^{n-1}(s^n, a^n) + \alpha_n \hat{q}^n(s^n, a^n), \tag{10}$$

where γ is a discount factor and α_n is a smoothing factor that might be called a stepsize (the equation has roots in stochastic optimization) or learning rate. Equation (10) is the core of "reinforcement learning."

We assume that when we are in state s^n and take action a^n that we have some way of simulating the transition to a state s'. There are two ways of doing this:

- Model-based - We assume we have the transition matrix P and then sample s' from the probability distribution $p(s'|s,a)$.

- Model-free - We assume we are observing a physical setting where we can simply observe the transition to state s' without a transition matrix.

Sutton and Barto named this algorithmic strategy "Q-learning" (after the notation). The appeal of the method is its sheer simplicity. In fact, they retained this style in their wildly popular book (Sutton & Barto 1998) which can be read by a high-school student.

Just as appealing is the wide applicability of both the model and algorithmic strategy. Contrast the core algorithmic step described by equations (9) - (10) to Bellman's equation which is the foundation of Markov decision processes, which requires solving

$$V_t(s) = \max_a \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V_{t+1}(s') \right), \tag{11}$$

for all states $s \in \mathcal{S}$. Equation (11) is executed by setting $V_{T+1}(s) = 0$ for all $s \in \mathcal{S}$, and then stepping backward $t = T, T-1, \ldots, 1$ (hence the reason that this is often called "backward dynamic programming"). In fact, this version was so trivial that the field focused on the stationary version which is written

$$V(s) = \max_a \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s') \right). \tag{12}$$

[Side note: The steady state version of Bellman's equation in (12) became the default version of Bellman's equation, which explains why the default notation for reinforcement learning does not index variables by time. By contrast, the default formulation for optimal control is finite time, and variables are indexed by time in the canonical model.]

Equation (12) requires solving a system of nonlinear equations to find $V(s)$, which proved to be the foundation for an array of papers with elegant algorithms, where one of the most important is

$$V^{n+1}(s) = \max_a \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^n(s') \right). \tag{13}$$

Equation (13) is known as value iteration (note the similarity with equation (11)) and is the basis of Q-learning (compare to equations (9)-(10)).

The problem with (11) is that it is far from trivial. In fact, it is quite rare that it can be computed due to the widely cited "curse of dimensionality." This typically refers to the fact that for most problems, the state s is a vector $s = (s_1, s_2, \ldots, s_K)$. Assuming that all the states are discrete, the number of states grows exponentially in K. It is for this reason that dynamic programming is widely criticized for suffering from the "curse of dimensionality." In fact, the curse of dimensionality is due purely to the use of lookup table representations of the value function (note that the canonical optimal control model does not do this).

In practice, this typically means that one-dimensional problems can be solved in under a minute; two-dimensional problems might take several minutes (but possibly up to an hour, depending on the dimensionality and the planning horizon); three dimensional problems easily take a week or a month; and four dimensional problems can take up to a year (or more).

In fact, there are actually three curses of dimensionality: the state space, the action space, and the outcome space. It is typically assumed that there is a discrete set of actions (think of the roads emanating from an intersection), but there are many problems where decisions are vectors (think of all the ways of assigning different taxis to passengers). Finally, there are random variables (call them W for now) which might also be a vector. For example, W might be the set of riders calling our taxi company for rides in a 15 minute period. Or, it might represent all the attributes of a customer clicking on ads (age, gender, location).

The most difficult computational challenge in equation (11) is finding the one-step transition matrix P with element $p(s'|s, a)$. This matrix measures $|\mathcal{S}| \times |\mathcal{S}| \times |\mathcal{A}|$ which may already be quite large. However, consider what it takes to compute just one element. To show this, we need to steal a bit of notation from the optimal control model, which is the *transition function* $f(x, u, w)$. Using the notation of dynamic programming, we would write this as $f(s, a, w)$. Assuming the transition function is known, the one-step transition matrix is computed using

$$p(s'|s, a) = \mathbb{E}_w\{\mathbb{1}_{\{s'=f(s,a,w)\}}\}. \tag{14}$$

This is not too hard if the random variable w is scalar, but there are many problems where w is a vector, in which case we encounter the third curse of dimensionality. There are many other problems where we do not even know the distribution of w (but have a way of observing outcomes).

Now return to the Q-learning equations (9) - (10). At no time are we enumerating all the states, although we do have to enumerate all the actions (and the states these actions lead to), which is perhaps a reason why reinforcement learning is always illustrated in the context of relatively small, discrete action spaces (think of the Chinese game of Go). Finally, we do not need to take an expectation over the random variable w; rather, we just simulate our way from state s to state s' using the transition function $f(s, a, w)$.
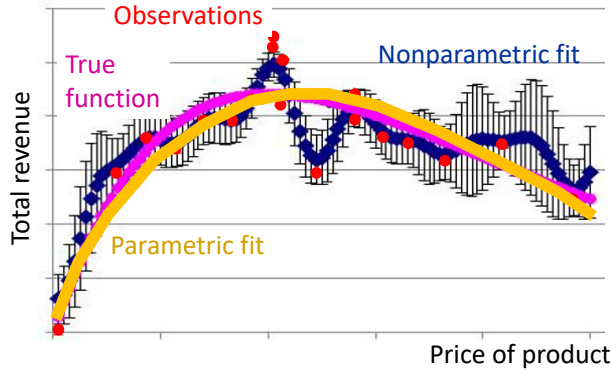
Figure 2: Nonparametric fit (blue line) vs. parametric fit (yellow line), compared to actual (purple line).

We are not out of the woods. We still have to estimate the value of being in state s and taking action a, captured by our Q-factors $\bar{Q}(s, a)$. If we use lookup tables, this means we need to estimate $\bar{Q}(s, a)$ for each state that we *might* visit, and each action that we *might* take, which means we are back to the curse of dimensionality. However, we can use other approximation strategies:

- Lookup tables with hierarchical beliefs - Here we use a family of lookup table models at different levels of aggregation.

- Parametric models, which might be linear (in the parameters) or nonlinear. We include shallow neural networks here. Parametric models transform the dimensionality of problems down to the dimensionality of the parameter vector, but we have to know the parametric form.

- Nonparametric models. Here we include kernel regression, locally parametric, and flexible architectures such as support vector machines and deep neural networks.

Not surprisingly, considerable attention has been devoted to different methods for approximating $\bar{Q}(s, a)$, with recent attention focusing on using deep neural networks. We will just note that the price of higher-dimensional architectures is that they come with the price of increased training (in fact, dramatically increased training). A deep neural network might easily require tens of millions of iterations, and yet still may not guarantee high quality solutions.

The challenge is illustrated in figure 2, where we have a set of observations (red dots). We try to fit the observations using a nonparametric model (the blue line) which overfits the data, which we believe is a smooth, concave (approximately quadratic) surface. We would get a reasonable fit of a quadratic function with no more than 10 data points, but any nonparametric model (such as a deep neural network) might require hundreds to thousands of data points (depending on the noise) to get a good fit.

3.3 A critique of the MDP modeling framework

For many years, the modeling framework of Markov decision processes lived within the MDP community which consisted primarily of applied probabilists, reflecting the limited applicability of the solution methods. Reinforcement learning, however, is a field that is exploding in popularity, while still clinging to the classical MDP modeling framework (see Lazaric (2019) for a typical example of this). What is happening, however, is that people doing computational work are adopting styles that overcome the limitations of the discrete MDP framework. For example, researchers will overcome the problem of computing the one-step transition matrix $p(s'|s,a)$ by saying that they will "simulate" the process. In practice, this means that they are using the transition function $f(s,a,w)$, which means that they have to simulate the random information w, without explicitly writing out $f(s,a,w)$ or the model of w. This introduces a confusing gap between the statement of the model and the software that captures and solves the model.

We offer the following criticisms of the classical MDP framework that the reinforcement learning community has adopted:

- The MDP/RL modeling framework models state *spaces*. The optimal control framework models state *variables*. We argue that the latter is much more useful, since it more clearly describes the actual variables of the problem. Consider a problem with discrete states (perhaps with K dimensions). The state space could then be written $\mathcal{S} = \mathcal{S}_1 \times \cdots \times \mathcal{S}_K$ which produces a set of discrete states that we can write $\{1, 2, \ldots, |\mathcal{S}|\}$. If we had a magical tool that could solve discrete Markov decision problems (remember we need to compute the one-step transition matrix), then we do not need to know anything about the state space $\mathcal{S}$, but this is rarely the case. Further, we make the case that just knowing that we have $|\mathcal{S}|$ states provides no information about the problem itself, while a list of the variables that make up the state variable (as is done in optimal control) will map directly to the software implementing the model.

- Similarly, the MDP/RL community talks about action *spaces*, while the controls community uses control *variables*. There is a wide range of problems that are described by discrete actions, where the action space is not too large. However, there are also many problems where actions are continuous, and often are vector valued. The notation of an "action space" $\mathcal{A}$ is simply not useful for vector-valued decisions/controls (the issue is the same as with state spaces).

- The MDP modeling framework does not explicitly model the exogenous information process w_t. Rather, it is buried in the one-step transition function $p(s'|s,a)$, as is seen in equation (14). In practical algorithms, we need to simulate the w_t process, so it helps to model the process explicitly. We would also argue that the model of w_t is a critical and challenging dimension of any sequential decision problem which is overlooked in the canonical MDP modeling framework.

- Transition functions, if known, are always computable, since we just have to compute them

for a single state, a single action, and a single observation of any exogenous information. We suspect that this is why the optimal control community adopted this notation. One-step transition matrices (or one-step transition kernels if the state variable is continuous) are almost never computable.

- There is no proper statement of the objective function, beyond the specification of the reward function $r(s, a)$. There is an implied objective similar to equation (4), but as we are going to see below, objective functions for sequential decision problems come in a variety of styles. Most important, in our view, is the need to state the objective in terms of optimizing over *policies*.

We suspect that the reason behind the sharp difference in styles between optimal control and Markov decision processes (adopted by reinforcement learning) is that the field of optimal control evolved from engineering, while Markov decision processes evolved out of mathematics. The adoption of the MDP framework by reinforcement learning (which grew out of computer science and is particularly popular with less-mathematical communities) is purely historical - it was easier to make the connection between the discrete mouse-in-a-maze problem to the language of discrete Markov decision processes than stochastic optimal control.

In section 4 below we are going to offer a framework that overcomes all of these limitations. This framework, however, closely parallels the framework widely used in optimal control, with a few relatively minor modifications (and one major one).

3.4 Bridging optimal control and reinforcement learning

We open our discussion by noting the remarkable difference between the canonical modeling framework for optimal control, which explicitly models state variables x_t, controls u_t, information w_t, and transition functions, and the canonical modeling framework for reinforcement learning (inherited from Markov decision processes) which uses constructs such as state spaces, action spaces, and one-step transition matrices. We will argue that the framework used in optimal control can be translated directly to software, whereas that used by reinforcement learning does not.

To illustrate this assertion, we note that optimal control and reinforcement learning are both addressing a sequential decision problem. In the notation of optimal control, we would write (focusing on discrete time settings):

$$(x_0, u_0, w_0, x_1, u_1, w_1, \ldots, x_T).$$

The reinforcement learning framework, on the other hand, never models anything comparable to the information variable w_t. In fact, the default setting is that we just observe the downstream state rather than modeling how we get there, but this is not universally true.

We also note that while the controls literature typically indexes variables by time, the RL community adopted the standard steady state model (see equation (12)) which means their variables are not indexed by time (or anything else). Instead, they view the system as evolving in steps (or iterations). For this reason, we are going to index variables by n (as in S^n).

In addition, the RL community does not explicitly model an exogenous information variable. Instead, they tend to assume when you are in a state s and take action a, you then "observe" the next state s'. However, any simulation in a reinforcement learning model requires creating a transition function which may (but not always) involve some random information that we call "w" (adopting, for the moment, the notation in the controls literature). This allows us to write the sequential decision problem

$$(S^0, a^0, (w^1), S^1, a^1, (w^2), S^2, \ldots, S^N).$$

We put the (w^n) in parentheses because the RL community does not explicitly model the w^n process. However, when running an RL simulation, the software will have to model this process, even if we are just observing the next state. We use w^{n+1} after taking action a^n simply because it is often the case that $S^{n+1} = w^{n+1}$.

We have found that the reinforcement learning community likes to start by stating a model in terms of the MDP formal model, but then revert to the framework of stochastic control. A good example is the presentation by Lazaric (2019); slide 22 presents the MDP formal model, but when the presentation turns to present an illustration (using a simple inventory problem), it turns to the style used in the optimal control community (see slide 29). Note that the presentation insists that the demand be stationary, which seems to be an effort to force it into the standard stationary model (see equation (12)). We use a much more complex inventory problem in this article, where we do not require stationarity (and which would not be required by the canonical optimal control framework).

So, we see that both optimal control and reinforcement learning are solving sequential decision problems, also known as Markov decision problems. Sequential decision problems (decision, information, decision, information, ...) span a truly vast range of applications, as noted in section 2. We suspect that this space is much broader than has been traditionally viewed within either of these two communities. This is not to say that all these problems can be solved with Q-learning or even any Bellman-based method, but below we will identify four classes of policies that span any approach that *might* be used for any sequential decision problems.

The optimal control literature has its origins in problems with continuous states and actions, although the mathematical model does not impose any restrictions beyond the basic structure of sequential decisions and information (for stochastic control problems). While optimal control is best known for the theory surrounding the structure of linear-quadratic regulation which produces the linear policy in (8), it should not be surprising that the controls community branched into more

general problems, requiring different solution strategies. These include:

- Approximating the cost-to-go function $J_t(x_t)$.

- Determining a decision now by optimizing over a horizon $t, \ldots, t+H$ using a presumably-known model of the system (which is not always available). This approach became known as *model predictive control*

- Specifying a parametric control law, which is typically linear in the parameters (following the style of (8)).

At the same time, the reinforcement learning community found that the performance of Q-learning (that is, equations (9)-(10)), despite the hype, did not match early hopes and expectations. In fact, just as the optimal controls community evolved different solution methods, the reinforcement learning community followed a similar path (the same statement can be made of a number of fields in stochastic optimization). This evolution is nicely documented by comparing the first edition of Sutton and Barto's *Reinforcement Learning* (Sutton & Barto 1998), which focuses exclusively on Q-learning, with the second edition (Sutton & Barto 2018), which covers methods such as Monte Carlo tree search, upper confidence bounding, and the policy gradient method.

We are going to next present (in section 4) a universal framework which is illustrated in section 5 on a series of problems in energy storage. Section 6 will then present four classes of policies that cover every method that has been proposed in the literature, which span all the variations currently in use in both the controls literature as well as the growing literature on reinforcement learning. We then return to the energy storage problems in section 7 and illustrate all four classes of policies (including a hybrid).

4 The universal modeling framework

We are going to present a universal modeling framework that covers all of the disciplines and application domains listed in section 2. The framework will end up posing an optimization problem that involves searching over *policies*, which are functions for making decisions. We will illustrate the framework on a simple inventory problem using the setting of controlling battery storage (a classical stochastic control problem).

In section 5 we will illustrate some key concepts by extending our energy storage application, focusing primarily on modeling state variables. Then, section 6 describes a general strategy for designing policies, which we are going to claim covers *every* solution approach proposed in the research literature (or used in practice). Thus, we will have a path to finding solutions to *any* problem (but these are rarely optimal).

Before starting, we make a few notes on notation:

- The controls community uses x_t for state, while the reinforcement learning community adopted the widely used notation S_t for state. We have used S_t partly because of the mnemonics (making it easier to remember), but largely because x_t conflicts with the notation for decisions adopted by the field of math programming, which is widely used.

- There are three standard notational systems for decisions: a for action (typically discrete), u for control (typically a low-dimensional, continuous vector), and x, which is the notation used by the entire math programming community, where x can be continuous or discrete, scalar or vector. We adopt x because of how widely it is used in math programming, and because it has been used in virtually every setting (binary, discrete, continuous, scalar or vector). It has also been adopted in the multi-armed bandit community in computer science.

- The controls community uses w_t which is (sadly) random at time t, whereas all other variables are known at time t. We prefer the style that every variable indexed by time t (or iteration n) is known at time t (or iteration n). For this reason, we use W_t for the exogenous information that first becomes known between $t-1$ and t, which means it is known at time t. (Similarly, W^n would be information that becomes known between iterations/observations $n-1$ and n.)

4.1 Dimensions of a sequential decision model

There are five elements to any sequential decision problem: state variables, decision variables, exogenous information variables, transition function, and objective function. We briefly describe each below, returning in section 4.2 to discuss state variables in more depth. The description below is adapted from Powell (2019).

State variables - The state S_t of the system at time t (we might say S^n after n iterations) is a function of history which contains all the information that is necessary and sufficient to compute costs/rewards, constraints, and any information needed by the transition function. The state S_t typically consists of a number of dimensions which we might write as $S_t = (S_{t1}, \ldots, S_{tK})$. This will be more meaningful when we illustrate it with an example below.

We distinguish between the initial state S_0 and the dynamic state S_t for $t > 0$. The initial state contains all deterministic parameters, initial values of any dynamic parameters, and initial beliefs about unknown parameters in the form of the parameters of probability distributions. The dynamic state S_t contains only information that is evolving over time.

In section 4.2, we will distinguish different classes of state variables, including physical state variables R_t (which might describe inventories or the location of a vehicle), other information I_t (which might capture prices, weather, or the humidity in a laboratory), and beliefs B_t (which

includes the parameters of probability distributions describing unobservable parameters). It is sometimes helpful to recognize that (R_t, I_t) capture everything that can be observed perfectly, while B_t represents distributions of anything that is uncertain.

Decision variables - We use x_t for decisions, where x_t may be binary (e.g. for a stopping problem), discrete (e.g. an element of a finite set), continuous (scalar or vector), integer vectors, and categorical (e.g. the attributes of a patient). In some applications x_t might have hundreds of thousands, or even millions, of *dimensions*, which makes the concept of "action spaces" fairly meaningless. We note that entire fields of research are sometimes distinguished by the nature of the decision variable.

We assume that decisions are made with a policy, which we might denote $X^\pi(S_t)$. We also assume that a decision $x_t = X^\pi(S_t)$ is feasible at time t. We let "π" carry the information about the type of function $f \in \mathcal{F}$ (for example, a linear model with specific explanatory variables, or a particular nonlinear model), and any tunable parameters $\theta \in \Theta^f$.

Exogenous information - We let W_t be any new information that first becomes known at time t (that is, between $t-1$ and t). This means any variable indexed by t is known at time t. When modeling specific variables, we use "hats" to indicate exogenous information. Thus, $\hat{D}_t$ could be the demand that arose between $t-1$ and t, or we could let $\hat{p}_t$ be the change in the price between $t-1$ and t. The exogenous information process may be stationary or nonstationary, purely exogenous or state (and possibly action) dependent.

As with decisions, the exogenous information W_t might be scalar, or it could have thousands to millions of *dimensions* (imagine the number of new customer requests for trips from zone i to zone j in an area that has 20,000 zones).

The distribution of W_{t+1} (given we are at time t) may be described by a known mathematical model, or we may depend on observations from an exogenous source (this is known as "data driven"). The exogenous information may depend on the current state and/or action, so we might write it as $W_{t+1}(S_t, x_t)$. We will suppress this notation moving forward, but with the understanding that we allow this behavior.

Transition function - We denote the transition function by

$$S_{t+1} = S^M(S_t, x_t, W_{t+1}), \tag{15}$$

where $S^M(\cdot)$ is also known by names such as system model, state equation, plant model, plant equation and transfer function. We have chosen not to use the standard notation $f(s, x, w)$ used universally by the controls community simply because the letter f is also widely used for "functions" in many settings. The alphabet is very limited and the letter f occupies a valuable piece of real-estate.

An important problem class in both optimal control and reinforcement learning arises when the transition function is unknown. This is sometimes referred to as "model-free dynamic programming." There are some classes of policies that do not need a transition function, both others do, introducing the dimension of trying to learn the transition function.

Objective functions - There are a number of ways to write objective functions in sequential decision problems. Our default notation is to let

$C_t(S_t, x_t) =$ the contribution of taking action x_t given the information in state S_t.

For now we are going to use the most common form of an objective function used in both dynamic programming (which includes reinforcement learning) and stochastic control, which is to maximize the expected sum of contributions:

$$\max_{\pi} \mathbb{E}_{S_0} \mathbb{E}_{W_1,\ldots,W_T|S_0} \left\{ \sum_{t=0}^{T} C_t(S_t, X_t^\pi(S_t)) | S_0 \right\},\tag{16}$$

where

$$S_{t+1} = S^M(S_t, X_t^\pi(S_t), W_{t+1}),\tag{17}$$

and where we are given a source of the exogenous information process

$$(S_0, W_1, W_2, \ldots, W_T).\tag{18}$$

We refer to equation (16) along with the state transition function (17) and exogenous information (18) as the *base model*. We revisit objective functions in section 4.3.

An important feature of our modeling framework is that we introduce the concept of a policy $X^\pi(S_t)$ when we describe decisions, and we search over policies in the objective function in equation (16), but we do not at this point specify what the policies might look like. Searching over policies is precisely what is meant by insisting that the control u_t in equation (4) be "$\mathcal{F}_t$-measurable." In section 6 we are going to make this much more concrete, and does not require mastering subtle concepts such as "measurability." All that is needed is the understanding that a policy depends on the state variable (measurability is guaranteed when this is the case).

In other words (and as promised), we have modeled the problem without specifying how we would solve them (that is, we have not specified how we are computing the policy). This follows our *"Model first, then solve"* approach. Contrast this with the Q-learning equations (9) - (10) which is basically an algorithm without a model, although the RL community would insist that the model is the canonical MDP framework given in section 3.2.

4.2 State variables

Our experience is that there is an almost universal misunderstanding of what is meant by a "state variable." Not surprisingly, interpretations of the term "state variable" vary between communities. An indication of the confusion can be traced to attempts to define state variables. For example, Bellman introduces state variables with "we have a physical system characterized at any stage by a small set of parameters, the *state variables*" (Bellman 1957). Puterman's now classic text introduces state variables with "At each decision epoch, the system occupies a *state*." (Puterman 2005)[p. 18] (in both cases, the italicized text was included in the original text). As of this writing, Wikipedia offers "A state variable is one of the set of variables that are used to describe the mathematical state of a dynamical system." Note that all three references use the word "state" in the definition of state variable.

It has also been our finding that most books in optimal control do, in fact, include proper definitions of a state variable (our experience is that this is the only field that does this). They all tend to say the same thing: a state variable x_t is all the information needed to model the system from time t onward.

Our only complaint about the standard definition used in optimal control books is that it is vague. The definition proposed in Powell (2020) (building on the definition in Powell (2011)) refines the basic definition with the following:

> A **state variable** is:
>
> a) **Policy-dependent version** A function of history that, combined with the exogenous information (and a policy), is necessary and sufficient to compute the decision function (the policy), the cost/contribution function, and the transition function.
> b) **Optimization version** A function of history that, combined with the exogenous information, is necessary and sufficient to compute the cost or contribution function, the constraints, and the transition function.

There are three types of information in S_t:

- The physical state, R_t, which in most (but not all) applications is the state variables that are being controlled. R_t may be a scalar, or a vector with element R_{ti} where i could be a type of resource (e.g. a blood type) or the amount of inventory at location i. Physical state variables typically appear in the constraints. We make a point of singling out physical states because of their importance in modeling resource allocation problems, where the "state of the system" is often (and mistakenly) equated with the physical state.

- Other information, I_t, which is any information that is known deterministically not included in R_t. The information state often evolves exogenously, but may be controlled or at least

influenced by decisions (e.g. selling a large number of shares may depress prices). Other information may appear in the objective function (such as prices), and the coefficients in the constraints.

- The belief state B_t, which contains distributional information about unknown parameters, where we can use frequentist or Bayesian belief models. These may come in the following styles:

  - Lookup tables - Here we have a set of discrete values $x \in \mathcal{X} = \{x_1, \ldots, x_M\}$, and we have a belief about a function (such as $f(x) = \mathbb{E}F(x, W)$) for each $x \in \mathcal{X}$.

  - Parametric belief models - We might assume that $\mathbb{E}F(x, W) = f(x|\theta)$ where the function $f(x|\theta)$ is known but where θ is unknown. We would then describe θ by a probability distribution.

  - Nonparametric belief models - These approximate a function at x by smoothing local information near x.

It is important to recognize that the belief state includes the parameters of a probability distribution describing unobservable parameters of the model. For example, B_t might be the mean and covariance matrix of a multivariate normal distribution, or a vector of probabilities $p^n = (p_k^n)_{k=1}^K$ where $p_k^n = Prob[\theta = \theta_k | S^n]$. [1]

We feel that a proper understanding of state variables opens up the use of the optimal control framework to span the entire set of communities and applications discussed in section 2.

4.3 Objective functions

Sequential decision problems are diverse, and this is reflected in the different types of objective functions that may be used. Our framework is insensitive to the choice of objective function, but they all still require optimizing over policies.

We begin by making the distinction between state-independent problems, and state-dependent problems. We let $F(x, W)$ denote a state-independent problem, where we assume that neither the objective function $F(x, W)$, nor any constraints, depends on dynamic information captured in the state variable. We let $C(S, x)$ capture state-dependent problems, where the objective function (and/or constraints) may depend on dynamic information.

---

[1] It is not unusual for people to overlook the need to include beliefs in the state variable. The RL tutorial Lazaric (2019) does this when it presents the multiarmed bandit problem, insisting that it does not have a state variable (see slide 49). In fact, any bandit problem is a sequential decision problem where the state variable is the belief (which can be Bayesian or frequentist). This has long been recognized by the probability community that has worked on bandit problems since the 1950's (see the seminal text DeGroot (1970)). Bellman's equation (using belief states) was fundamental to the development of Gittins indices in Gittins & Jones (1974) (see Gittins et al. (2011) for a nice introduction to this rich area of research). It was the concept of Gittins indices that laid the foundation for upper confidence bounding, which is just a different form of index policy.

Throughout we assume problems are formulated over finite time horizons. This is the most standard approach in optimal control, whereas the reinforcement learning community adopted the style of Markov decision processes to model problems over infinite time horizons. We suspect that the difference reflects the history of optimal control, which is based on solving real engineering problems, and Markov decision processes, with its roots in mathematics and stylized problems.

In addition to the issue of state-dependency, we make the distinction between optimizing the cumulative reward versus the final reward. When we combine state dependency and the issue of final vs. cumulative reward, we obtain four objective functions. We present these in the order: 1) State-independent, final reward, 2) state-independent, cumulative reward, 3) state-dependent, cumulative reward, and 4) state-dependent, final reward (the last class is the most subtle).

State-independent functions These are pure learning problems, where the *problem* does not depend on information in the state variable. The only state variable is the belief about an unknown function $\mathbb{E}_W F(x, W)$.

1) **Final reward** This is the classical stochastic search problem. Here we go through a learning/training process to find a final design/decision $x^{\pi,N}$, where π is our search policy (or algorithm), and N is the budget. We then have to test the performance of the policy by simulating $\widehat{W}$ using

$$\max_{\pi} \mathbb{E}_{S^0} \mathbb{E}_{W^1,...,W^N | S^0} \mathbb{E}_{\widehat{W} | S^0} F(x^{\pi,N}, \widehat{W}), \tag{19}$$

where $x^{\pi,N}$ depends on S^0 and the experiments $W^1, \ldots, W^N$, and where $\widehat{W}$ represents the process of testing the design $x^{\pi,N}$.

2) **Cumulative reward** This describes problems where we have to learn in the field, which means that we have to optimize the sum of the rewards we earn, eliminating the need for a final testing pass. This objective is written

$$\max_{\pi} \mathbb{E}_{S^0} \mathbb{E}_{W^1,...,W^N | S^0} \sum_{n=0}^{N-1} F(X^{\pi}(S^n), W^{n+1}). \tag{20}$$

State-dependent functions This describes the massive universe of problems where the objective and/or the constraints depend on the state variable which may or may not be controllable.

3) **Cumulative reward** This is the version of the objective function that is most widely used in stochastic optimal control (as well as Markov decision processes). We switch back to time-indexing here since these problems are often evolving over time (but not always). We write the contribution in the form $C(S_t, x_t, W_{t+1})$ to help with the comparison to $F(x, W)$.

$$\max_{\pi} \mathbb{E}_{S_0} \mathbb{E}_{W_1,...,W_T | S_0} \left\{ \sum_{t=0}^{T} C(S_t, X^{\pi}(S_t), W_{t+1}) | S_0 \right\}. \tag{21}$$

4) Final reward This is the objective function that describes optimization algorithms (represented as π^{lrn}) optimizing a time-staged, state-dependent objective. This is the objective that should be used when finding the best algorithm for a dynamic program/stochastic control problem, yet has been almost universally overlooked as a sequential decision problem. The objective is given by

$$\max_{\pi^{lrn}} \mathbb{E}_{S^0} \mathbb{E}_{W^1,\ldots,W^N|S^0}^{\pi^{lrn}} \mathbb{E}_{S|S^0}^{\pi^{imp}} \mathbb{E}_{\widehat{W}|S^0} C(S, X^{\pi^{imp}}(S|\theta^{\pi^{imp}}), \widehat{W}). \tag{22}$$

where π^{lrn} is the learning policy (or algorithm), while π^{imp} is the implementation policy that we are learning through π^{lrn}. We note that we use the learning policy π^{lrn} to learn the parameters $\theta^{\pi^{imp}}$ that govern the behavior of the implementation policy.

There are many problems that require more complex objective functions such as the best (or worst) performance in a time period, across all time periods. In these settings we cannot simply sum the contributions across time periods (or iterations). For this purpose, we introduce the operator ρ which takes as input the entire sequence of contributions. We would write our objective function as

$$\max_{\pi} \mathbb{E}_{S^0} \mathbb{E}_{W^1,\ldots,W^N|S^0} \ \rho\big(C_0(S_0, X^\pi(S_0), W_1), C_2(S_2, X^\pi(S_2), W_3), \ldots, C_t(S_t, X_t^\pi(S_t), W_{t+1}), \ldots\big). \tag{23}$$

The objective in (23), through creative use of the operator ρ, subsumes all four objectives (19) - (22). However, we feel that generality comes at a cost of clarity.

The controls community, while also sharing an interest in risk, is also interested in stability, an issue that is important in settings such as controlling aircraft and rockets. While we do not address the specific issue of designing policies to handle stability, we make the case that the problem of searching over policies remains the same; all that has changed is the metric.

All of these objectives can be written in the form of *regret* which measures the difference between the solution we obtain and the best possible. Regret is popular in the learning community where we compare against the solution that assumes perfect information. A comparable strategy compares the performance of a policy against what can be achieved with perfect information about the future (widely known as a posterior bound).

4.4   Notes

It is useful to list some similarities (and differences) between our modeling framework and that used in stochastic optimal control:

1) The optimal control framework includes all five elements, although we lay these out more explicitly.

2) We use a richer understanding of state variables, which means that we can apply our framework to a much wider range of problems than has traditionally been considered in the optimal control literature. In particular, *all* the fields and problem areas in section 2 fit this framework, which means we would say that all of these are "optimal control problems."

3) The stochastic control modelling framework uses w_t as the information that will arrive between t and $t + 1$, which means it is random at time t. We let W_t be the information that arrives between $t - 1$ and t, which means it is known at time t. This means we write our transition as

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

This notation makes it explicitly clear that W_{t+1} is not known when we determine decision x_t.

4) We recognize a wider range of objective functions, which expands the problem classes to offline and online applications, active learning (bandit) problems, and hybrids.

5) We formulate the optimization problem in terms of optimizing over policies, without prejudging the classes of policies. We describe four classes of policies in section 6 that we claim are universal: they cover all the strategies that have been proposed or used in practice. This also opens the door to creating hybrids that combine two or more classes of policies.

The first four items are relatively minor, highlighting our belief that stochastic control is fundamentally the most sound of all the modeling frameworks used by any of the communities listed in section 2. However, the fifth item is a significant transition from how sequential decision problems are approached today.

Many have found that Q-learning often does not work well. In fact, Q-learning, as with all approximate dynamic programming algorithms, tend to work well only on a fairly small set of problems. Our experience is that approximate dynamic programming (Q-learning is a form of approximate dynamic programming) tends to work well when we can exploit the structure of the value function. For example, ADP has been very successful with some very complex, high-dimensional problems in fleet management (see Simão et al. (2009) and Bouzaiene-Ayari et al. (2016)) where the value functions were convex. However, vanilla approximation strategies (e.g. using simple linear models for value function approximations) can work very poorly even on small inventory problems (see Jiang et al. (2014) for a summary of experiments which compare results against a rigorous benchmark). Furthermore, as we will see in section 6 below, there are a range of policies that do not depend on value functions that are natural choices for many applications.

5   Energy storage illustration

We are going to illustrate our modeling framework using the energy system depicted in figure 3, which consists of a wind farm (where energy is free but with high variability in supply), the grid (which
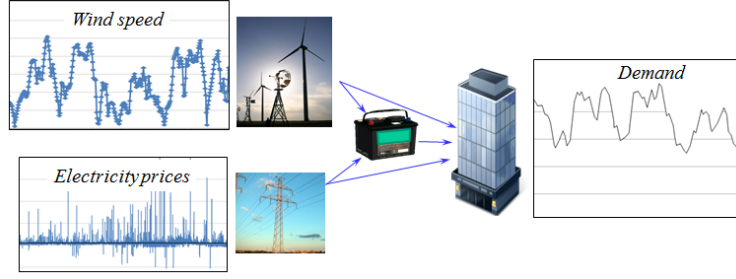
Figure 3: Energy system consisting of wind farms, grid, market, and battery storage.

has unlimited supply but highly stochastic prices), a market (which exhibits very time-dependent, although relatively predictable, demands), and an energy storage device (we will assume it is a battery). While small, this rich system introduces a variety of modeling and algorithmic challenges.

We are going to demonstrate how to model this problem, starting with a simple model and then expanding to illustrate some modeling devices. We will translate each variation into the five core components: state variables, decision variables, exogenous information variables, transition function, and objective function.

5.1 A basic energy storage problem

State variables State $S_t = (R_t, D_t, E_t, p_t)$ where

$$
\begin{aligned}
R_t &= \text{Energy in the battery at time } t, \\
E_t &= \text{Power being produced by the wind farms at time } t, \\
D_t &= \text{Demand for power at time } t, \\
p_t &= \text{Price of energy on the grid at time } t.
\end{aligned}
$$

Note that it is necessary to go through the rest of the model to determine which variables are needed to compute the objective function, constraints, and transition function.

Decision variables $x_t = (x_t^{GB}, x_t^{GD}, x_t^{EB}, x_t^{ED}, x_t^{BD})$ where

$$
\begin{aligned}
x_t^{GB} &= \text{Flow of energy from grid to battery } (x_t^{GB} > 0) \text{ or back } (x_t^{GB} < 0), \\
x_t^{GD} &= \text{Flow of energy from grid to demand}, \\
x_t^{EB} &= \text{Flow of energy from wind farm to battery}, \\
x_t^{ED} &= \text{Flow of energy from wind farm to demand}, \\
x_t^{BD} &= \text{Flow of energy from battery to demand}.
\end{aligned}
$$

25

These decision variables have to obey the constraints:

$$x_t^{EB} + x_t^{ED} \leq E_t, \tag{24}$$

$$x_t^{GD} + x_t^{BD} + x_t^{ED} = D_t, \tag{25}$$

$$x_t^{BD} \leq R_t, \tag{26}$$

$$x_t^{GD}, x_t^{EB}, x_t^{ED}, x_t^{BD} \geq 0. \tag{27}$$

Finally, we introduce the policy (function) $X^\pi(S_t)$ that will return a feasible vector x_t. We defer to later the challenge of designing a good policy.

Exogenous information variables $W_{t+1} = (\hat{E}_{t+1}, \hat{D}_{t+1}, \hat{p}_{t+1})$, where

$$\hat{E}_{t+1} = \text{The change in the power from the wind farm between } t \text{ and } t + 1,$$

$$\hat{D}_{t+1} = \text{The change in the demand between } t \text{ and } t + 1,$$

$$\hat{p}_{t+1} = \text{The price charged at time } t + 1 \text{ as reported by the grid.}$$

We note that the first two exogenous information variables are defined as changes in values, while the last (price) is reported directly from an exogenous source.

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

$$R_{t+1} = R_t + \eta(x_t^{GB} + x_t^{EB} - x_t^{BD}), \tag{28}$$

$$E_{t+1} = E_t + \hat{E}_{t+1}, \tag{29}$$

$$D_{t+1} = D_t + \hat{D}_{t+1}, \tag{30}$$

$$p_{t+1} = \hat{p}_{t+1}. \tag{31}$$

Note that we have illustrated here a controllable transition (28), two where the exogenous information is represented as the change in a process (equations (29) and (30)), and one where we directly observe the updated price (31). This means that the processes E_t and D_t are first-order Markov chains (assuming that $\hat{E}_t$ and $\hat{D}_t$ are independent across time), while the price process would be described as "model free" or "data driven" since we are not assuming that we have a mathematical model of the price process.

Objective function We wish to find a policy $X^\pi(S_t)$ that solves

$$\max_\pi \mathbb{E}_{S_0} \mathbb{E}_{W_1,\dots,W_T|S_0} \left\{ \sum_{t=0}^{T} C(S_t, X^\pi(S_t))|S_0 \right\},$$

where $S_{t+1} = S^M(S_t, x_t = X^\pi(S_t), W_{t+1})$ and where we are given an information process

$$(S_0, W_1, W_2, \dots, W_T).$$

Normally, we would transition at this point to describe how we are modeling the uncertainty in the information process $(S_0, W_1, W_2, \ldots, W_T)$, and then describe how to design policies. For compactness, we are going to skip these steps now, and instead illustrate how to model a few problem variations that can often cause confusion.

5.2 With a time-series price model

We are now going to make a single change to the model above. Instead of assuming that prices p_t are provided exogenously, we are going to assume we can model them using a time series model given by

$$p_{t+1} = \theta_0 p_t + \theta_1 p_{t-1} + \theta_2 p_{t-2} + \varepsilon_{t+1}. \tag{32}$$

A common mistake is to say that p_t is the "state" of the price process, and then observe that it is no longer Markovian (it would be called "history dependent"), but "it can be made Markovian by expanding the state variable," which would be done by including p_{t-1} and p_{t-2} (see Cinlar (2011) for an example of this). According to our definition of a state variable, the state is all the information needed to model the process from time t onward, which means that the state of our price process is (p_t, p_{t-1}, p_{t-2}). This means our system state variable is now

$$S_t = \big((R_t, D_t, E_t), (p_t, p_{t-1}, p_{t-2}))\big).$$

We then have to modify our transition function so that the "price state variable" at time $t+1$ becomes (p_{t+1}, p_t, p_{t-1}).

5.3 With passive learning

We implicitly assumed that our price process in equation (32) was governed by a model where the coefficients $\theta = (\theta_0, \theta_1, \theta_2)$ were known. Now assume that the price p_{t+1} depends on prices over the last three time periods, which means we would write

$$p_{t+1} = \bar{\theta}_{t0} p_t + \bar{\theta}_{t1} p_{t-1} + \bar{\theta}_{t2} p_{t-2} + \varepsilon_{t+1}. \tag{33}$$

Here, we have to adaptively update our estimate $\bar{\theta}_t$ which we can do using recursive least squares. To do this, let

$$\bar{p}_t = (p_t, p_{t-2}, p_{t-2})^T,$$
$$\bar{F}_t(\bar{p}_t | \bar{\theta}_t) = (\bar{p}_t)^T \bar{\theta}_t$$

We perform the updating using a standard set of updating equations given by

$$\bar{\theta}_{t+1} = \bar{\theta}_t + \frac{1}{\gamma_t} M_t \bar{p}_t \varepsilon_{t+1}, \tag{34}$$

$$\varepsilon_{t+1} = \bar{F}_t(\bar{p}_t|\bar{\theta}_t) - p_{t+1}, \tag{35}$$

$$M_{t+1} = M_t - \frac{1}{\gamma_t} M_t(\bar{p}_t)(\bar{p}_t)^T M_t, \tag{36}$$

$$\gamma_t = 1 - (\bar{p}_t)^T M_t \bar{p}_t. \tag{37}$$

To compute these equations, we need the three-element vector $\bar{\theta}_t$ and the 3×3 matrix M_t. These then need to be added to our state variable, giving us

$$S_t = \big((R_t, D_t, E_t), (p_t, p_{t-1}, p_{t-2}), (\bar{\theta}_t, M_t)\big).$$

We then have to include equations (34) - (37) in our transition function.

5.4   With active learning

We can further generalize our model by assuming that our decision x_t^{GB} to buy or sell energy from or to the grid can have an impact on prices. We might propose a modified price model given by

$$p_{t+1} = \bar{\theta}_{t0} p_t + \bar{\theta}_{t1} p_{t-1} + \bar{\theta}_{t2} p_{t-2} + \bar{\theta}_{t3} x_t^{GB} + \varepsilon_{t+1}. \tag{38}$$

All we have done is introduce a single term $\bar{\theta}_{t3} x_t^{GB}$ to our price model. Assuming that $\theta_3 > 0$, this model implies that purchasing power from the grid ($x_t^{GB} > 0$) will increase grid prices, while selling power back to the grid ($x_t^{GB} < 0$) decreases prices. This means that purchasing a lot of power from the grid (for example) means we are more likely to observe higher prices, which may assist the process of learning θ. When decisions control or influence what we observe, then this is an example of *active learning*.

This change in our price model does not affect the state variable from the previous model, aside from adding one more element to $\bar{\theta}_t$, with the required changes to the matrix M_t. The change will, however, have an impact on the policy. It is easier to learn θ_{t3} by varying x_t^{GB} over a wide range, which means trying values of x_t^{GB} that do not appear to be optimal given our current estimate of the vector $\bar{\theta}_t$. Making decisions partly just to learn (to make better decisions in the future) is the essence of *active learning*, best known in the field of multiarmed bandit problems.

5.5   With rolling forecasts

Forecasting is such a routine activity in operational problems, it may come as a surprise that we have been modelling these problems incorrectly.

Assume we have a forecast $f_{t,t+1}^E$ of the energy E_{t+1} from wind, which means

$$E_{t+1} = f_{t,t+1}^E + \varepsilon_{t+1,1}, \tag{39}$$

where $\varepsilon_{t+1,1} \sim N(0, \sigma_\varepsilon^2)$ is the random variable capturing the one-period-ahead error in the forecast.

Equation (39) effectively replaces equation (29) in the transition function for the base model. However, it introduces a new variable, the forecast $f_{t,t+1}^E$, which must now be added to the state variable. This means we now need a transition equation to describe how $f_{t,t+1}^E$ evolves over time. We do this by using a two-period-ahead forecast, $f_{t,t+2}^E$, which is basically a forecast of $f_{t+1,t+2}^E$, plus an error, giving us

$$f_{t+1,t+2}^E = f_{t,t+2}^E + \varepsilon_{t+1,2}, \tag{40}$$

where $\varepsilon_{t+1,2} \sim N(0, 2\sigma_\varepsilon^2)$ is the two-period-ahead error (we are assuming that the variance in a forecast increases linearly with time). Now we have to put $f_{t,t+2}^E$ in the state variable, which generates a new transition equation. This generalizes to

$$f_{t+1,t'}^E = f_{t,t'}^E + \varepsilon_{t+1,t'-t}, \tag{41}$$

where $\varepsilon_{t+1,t'-t} \sim N(0, (t'-t)\sigma_\varepsilon^2)$.

This stops, of course, when we hit the planning horizon H. This means that we now have to add

$$f_t^E = (f_{tt'}^E)_{t'=t+1}^{t+H}$$

to the state variable, with the transition equations (41) for $t' = t+1, \ldots, t+H$. Combined with the learning statistics, our state variable is now

$$S_t = \big((R_t, D_t, E_t), (p_t, p_{t-1}, p_{t-2}), (\bar{\theta}_t, M_t), f_t^E\big).$$

It is useful to note that we have a nice illustration of the three elements of our state variable:

$$
\begin{aligned}
(R_t, D_t, E_t) \;&=\; \text{The physical state variables (note that they all appear in the right hand side of} \\
&\quad\; \text{constraints (24)-(27)),} \\
(p_t, p_{t-1}, p_{t-2}) \;&=\; \text{other information,} \\
((\bar{\theta}_t, M_t), f_t^E) \;&=\; \text{the belief state, since these parameters determine the distribution of belief about} \\
&\quad\; \text{variables that are not known perfectly.}
\end{aligned}
$$

5.6 Remarks

We note that all the models illustrated in this section are sequential decision problems, which means that all of them can be described as either stochastic control problems, or reinforcement learning problems. This is true whether state variables or decision/control variables are scalar or vector, discrete or continuous (or mixed). We have, however, assume that time is either discrete or discretized.

Energy storage is a form of inventory problem, which is the original stochastic control problem used by Bellman to motivate his work on dynamic programming (Bellman et al. 1955), and is even used today by the reinforcement learning community (Lazaric 2019). However, we have never seen the variations that we illustrated here solved by any of these communities.

In section 6 we are going to present four classes of policies, and then illustrate, in section 7, that each of the four classes (including a hybrid) can be applied to the full range of these energy storage problems. We are then going to show that both communities (optimal control and reinforcement learning) use methods that are drawn from each of the four classes, but apparently without an awareness that these are instances in broader classes, that can be used to solve complex problems.

6 Designing policies

There are two fundamental strategies for creating policies:

Policy search - Here we use any of the objective functions (19) - (23) to search within a family of functions to find the policy that works best. This means we have to a) find a class of function and b) tune any parameters. The challenge is finding the right family, and then performing the tuning (which can be hard).

Lookahead approximations - Alternatively, we can construct policies by approximating the impact of a decision now on the future. The challenge here is designing and computing the approximation of the future (this is also hard).

Either of these approaches can yield optimal policies, although in practice this is rare. Below we show that each of these strategies can be further divided into two classes, creating four (meta)classes of policies for making decisions. We make the claim that these are universal, which is to say that *any* solution approach to any sequential decision problem will use a policy drawn from one of these four classes, or a hybrid of two or more classes.

6.1 Policy search

Policy search involves tuning and comparing policies using the objective functions (19) - (23) so that they behave well when averaged over a set of sample paths. Assume that we have a class of functions $\mathcal{F}$, where for each function $f \in \mathcal{F}$, there is a parameter vector $\theta \in \Theta^f$ that controls its behavior. Let $X^\pi(S_t|\theta)$ be a function in class $f \in \mathcal{F}$ parameterized by $\theta \in \Theta^f$, where $\pi = (f, \theta)$, $f \in \mathcal{F}, \theta \in \Theta^f$. Policy search involves finding the best policy using

$$\max_{\pi \in (\mathcal{F}, \Theta^f)} \mathbb{E}_{S_0} \mathbb{E}_{W_1, \ldots, W_T | S_0} \left\{ \sum_{t=0}^{T} C(S_t, X^\pi(S_t)) | S_0 \right\}. \tag{42}$$

In special cases, this can produce an optimal policy, as we saw for the case of linear-quadratic regulation (see equation (8)).

Since we can rarely find optimal policies using (42), we have identified two sub-classes within the policy search class:

Policy function approximations (PFAs) - Policy function approximations can be lookup tables, parametric or nonparametric functions, but the most common are parametric functions. This could be a linear function such as

$$X^\pi(S_t|\theta) = \theta_0 + \theta_1 \phi_1(S_t) + \theta_2 \phi_2(S_t) + \ldots,$$

which parallels the linear control law in equation (8) (these are also known as "affine policies"). We might also use a nonlinear function such as an order-up-to inventory policy, a logistics curve, or a neural network. Typically there is no guarantee that a PFA is in the optimal class of policies. Instead, we search for the best performance within a class.

Cost function approximations (CFAs) - A CFA is

$$X^\pi(S_t|\theta) = \arg \max_{x \in \mathcal{X}_t^\pi(\theta)} \bar{C}_t^\pi(S_t, x|\theta),$$

where $\bar{C}_t^\pi(S_t, x|\theta)$ is a parametrically modified cost function, subject to a parametrically modified set of constraints. A popular example known to the computer science community is interval estimation where a discrete alternative $x \in \mathcal{X} = \{x_1, \ldots, x_M\}$ is chosen which maximizes

$$X^{IE}(S^n|\theta^{IE}) = \arg \max_{x \in \mathcal{X}} \left(\bar{\mu}_x^n + \theta^{IE} \bar{\sigma}_x^n \right)$$

where $\bar{\mu}_x^n$ is the current estimate of $\mathbb{E}_W F(x, W)$ after n experiments, and where $\bar{\sigma}_x^n$ is the standard deviation of the statistic $\bar{\mu}_x^n$. Here, θ^{IE} is a parameter that has to be tuned.

CFAs are widely used for solving large scale problems such as scheduling an airline or planning a supply chain. For example, we might introduce slack into a scheduling problem, or buffer stocks for an inventory problem.

Policy search is best suited when the policy has clear structure, such as inserting slack in an airline schedule, or selling a stock when the price goes over some limit. Neural networks have become popular recently because they assume no structure, but the price of this generality is that extensive tuning is needed. We urge caution with the use of high-dimensional architectures such as neural networks. There are many problems where we expect the policy to exhibit structure, such as increasing the dosage of a drug with the weight of a patient, or setting the bid price of a stock as a function of market indicators. Neural networks do not offer these guarantees, and would require a tremendous amount of training to produce this behavior.

6.2 Lookahead approximations

Just as we can, in theory, find an optimal policy using policy search, we can also find an optimal policy by modeling the downstream impact of a decision made now on the future. This can be written

$$X_t^*(S_t) = \arg\max_{x_t} \left(C(S_t, x_t) + \mathbb{E} \left\{ \max_\pi \mathbb{E} \left\{ \sum_{t'=t+1}^{T} C(S_{t'}, X_{t'}^\pi(S_{t'})) \middle| S_{t+1} \right\} \middle| S_t, x_t \right\} \right). \tag{43}$$

Equation (43) is daunting, but can be parsed in the context of a decision tree with discrete actions and discrete random outcomes (see figure 4). The states $S_{t'}$ correspond to the square nodes in the decision tree. The state S_t is the initial node, and the actions x_t are the initial actions. The first expectation is over the first set of random outcomes W_{t+1} (out of the outcome nodes resulting from each decision x_t). The imbedded policy π is the choice of decision for *each* decision node (state) over the horizon $t + 1, \ldots, T$. The second expectation is over $W_{t+2}, \ldots, W_T$.

In practice, a stochastic lookahead policy is generally impossible to compute (decision trees grow exponentially). There are two broad strategies for approximating the lookahead model:

Value function approximations (VFAs) - Our first approach is to replace the entire term capturing the future in (43) with an approximation of the value function (the controls community uses the term cost-to-go function). We can do this in two ways. The first is to replace the function starting at S_{t+1} with a value function $V_{t+1}(S_{t+1})$ giving us

$$X_t^{VFA}(S_t) = \arg\max_{x_t} \left(C(S_t, x_t) + \mathbb{E} \left\{ \bar{V}_{t+1}(S_{t+1}) | S_t \right\} \right) \tag{44}$$

where $S_{t+1} = S^M(S_t, x_t, W_{t+1})$, and where the expectation is over W_{t+1} conditioned on S_t (some write the conditioning as dependent on S_t and x_t). Since we generally cannot compute $V_{t+1}(S_{t+1})$, we can use various machine learning methods to replace it with some sort of approximation $\bar{V}_{t+1}(S_{t+1})$ called the value function approximation.

32

$$X_t^*(S_t) = \arg\max_{x_t} \left(C(S_t, x_t) + \mathbb{E} \left\{ \max_{\pi \in \Pi} \left\{ \mathbb{E} \sum_{t'=t+1}^{T} C(S_{t'}, X_{t'}^\pi(S_{t'})) \,|\, S_{t+1} \right\} \,|\, S_t, x_t \right\} \right)$$
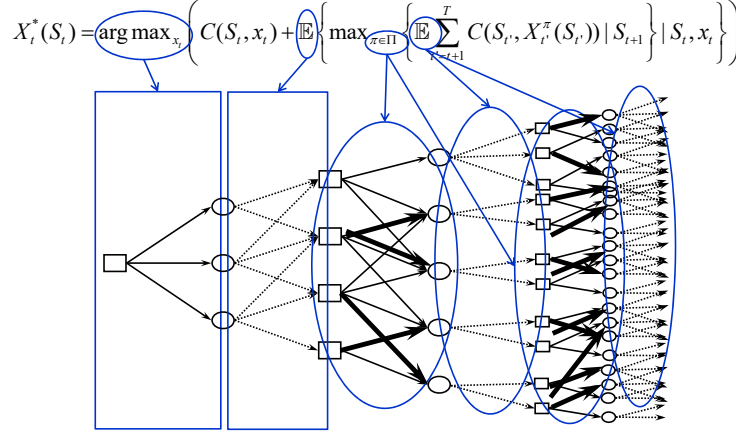
Figure 4: Relationship between the stochastic lookahead policy and a decision tree, showing initial decision, initial expectation, and then the decisions made for each state in the future (which is the lookahead policy $\tilde{\pi}$).

The second way is to approximate the function around the post-decision state S_t^x (this is the state immediately after a decision is made), which eliminates the expectation (44), giving us

$$X_t^{VFA}(S_t) = \arg\max_{x_t} \left(C(S_t, x_t) + \bar{V}_t^x(S_t^x) \right). \tag{45}$$

The benefit of using the post-decision value function approximation is that it eliminates the expectation from within the max operator. This has proven to be especially useful for problems where x_t is a vector, and $V_t^x(S_t^x)$ is a convex function of S_t^x.

There is by now an extensive literature on the use of value function approximations that have evolved under names such as heuristic dynamic programming (Si et al. 2004), neuro-dynamic programming (Bertsekas & Tsitsiklis 1996), adaptive dynamic programming (Murray et al. (2002), Lewis & Vrabie (2009)), approximate dynamic programming (Powell 2011), and reinforcement learning (Sutton & Barto 1998). While the use of value function approximations has tremendous appeal, it is no panacea. Our experience is that this approach works best only when we can exploit problem structure.

Direct lookahead (DLAs) There are many problems where it is just not possible to compute sufficiently accurate VFAs. When all else fails, we have to resort to a direct lookahead, where we replace the lookahead expectation and optimization in (43) with an *approximate lookahead model*.

The most widely used approximation strategy is to use a deterministic lookahead, often associated with model predictive control, although it is more accurate to refer to any policy based on solving a lookahead model as model predictive control. We can create an approximate

(stochastic) lookahead model that we represent as the following sequence

$$(S_t, x_t, \tilde{W}_{t,t+1}, \tilde{S}_{t,t+1}, \tilde{x}_{t,t+1}, \ldots, \tilde{W}_{tt'}, \tilde{S}_{tt'}, \tilde{x}_{tt'}, \tilde{W}_{t,t'+1}, \ldots).$$

We use tilde-variables to indicate variables within the lookahead model. Each tilde-variable is indexed by t (the time at which the lookahead model is being formed) and t' (the time within the lookahead model). Our lookahead policy might then be written

$$X_t^{DLA}(S_t) = \arg\max_{x_t} \left(C(S_t, x_t) + \tilde{E} \left\{ \max_{\tilde{\pi}} \tilde{E} \left\{ \sum_{t'=t+1}^{T} C(\tilde{S}_{tt'}, \tilde{X}^{\tilde{\pi}}(\tilde{S}_{tt'})) | \tilde{S}_{t,t+1} \right\} | S_t, x_t \right\} \right). \quad (46)$$

Typically the approximate expectations $\tilde{E}$ are computed using Monte Carlo sampling, although we can use a deterministic forecast. The real challenge is the lookahead policy $\tilde{X}^{\tilde{\pi}}(\tilde{S}_{tt'})$ which may take any form. This policy is also known as a "rollout policy" where it is used in combinatorial optimization (Bertsekas et al. 1997), and Monte Carlo tree search (Chang et al. (2005), Coulom (2007), Browne et al. (2012)).

One possibility for the lookahead policy is to use a simpler parameterized policy that we might write $\tilde{X}^{\tilde{\pi}}(\tilde{S}_{tt'}|\tilde{\theta})$. In this case, the $\max_{\tilde{\pi}}$ operator would be replaced with $\max_{\tilde{\theta}}$, but even this simpler problem means that we are finding the best parameter $\tilde{\theta}$ for each state $\tilde{S}_{t,t+1}$, which means we are really looking for a function $\tilde{\theta}(s)$ where $s = \tilde{S}_{t,t+1}$. A simpler alternative would be to fix a single parameter θ which means we now have a parameterized lookahead policy given by

$$X_t^{DLA}(S_t|\theta) = \arg\max_{x_t} \left(C(S_t, x_t) + \tilde{E} \left\{ \tilde{E} \left\{ \sum_{t'=t+1}^{T} C(\tilde{S}_{tt'}, \tilde{X}^{\tilde{\pi}}(\tilde{S}_{tt'}|\theta)) | \tilde{S}_{t,t+1} \right\} | S_t, x_t \right\} \right). \quad (47)$$

This version no longer has the imbedded $\max_{\tilde{\theta}}$, but we still have to tune θ in the policy $X_t^{DLA}(S_t|\theta)$.

Another strategy for computing a stochastic lookahead policy is to use Monte Carlo tree search, a term coined by (Coulom 2007) but first proposed in (Chang et al. 2005) (see Browne et al. (2012) for a tutorial in the context of deterministic problems). This strategy searches forward in time, using methods to limit the full enumeration of the tree. Monte Carlo tree search gained prominence from the role it played in creating AlphaGo for playing the Chinese game of Go, which was the first system to beat world class Go players (see Fu (2017) for a nice review of the history of MCTS and AlphaGo).

It is important to emphasize that designing a policy using a stochastic lookahead (even a simplified stochastic lookahead) means solving a stochastic optimization problem within the
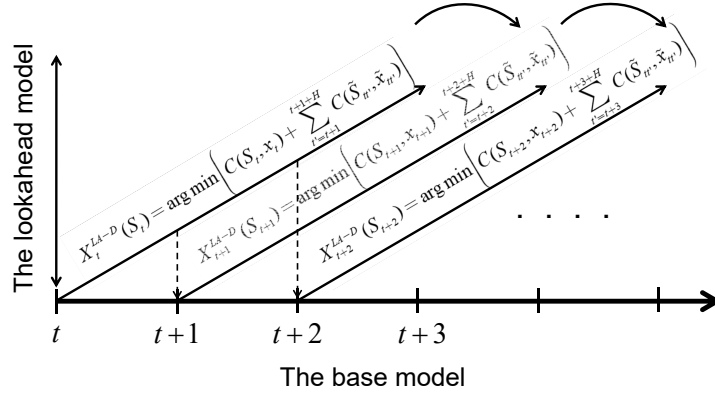
Figure 5: Illustration of simulating a lookahead policy using a deterministic lookahead model.

policy. Recall that our stochastic optimization problem is the base model given by any of the objective functions described earlier (equations (19) - (23)). Equation (47) represents the simplified stochastic optimization problem, which has to be solved at each time period.

Figure 5 depicts the process of simulating a direct lookahead (the figure uses a deterministic lookahead, but the same process would be used with any direct lookahead). This is what is needed to do any parameter tuning for the DLA. Not surprisingly, stochastic lookaheads can be computationally difficult to solve, which makes it particularly difficult to run simulations to do parameter tuning.

6.3 Hybrid policies

A byproduct of identifying the four classes of policies is the ability to create hybrids that combine two or more classes. Some examples include:

- Lookahead policies plus VFAs - We can do an H-period lookahead, and then terminate with a value function approximation. This will likely simplify the task of coming up with a good value function approximation, while also providing better results for the same horizon (allowing us to shorten the horizon for the tree search).

- Value function approximations with parameter tuning. Imagine that we create a VFA-based policy that looks like

$$X^{VFA}(S_t|\theta) = \arg\max_x \left(C(S_t, x) + \sum_{f \in \mathcal{F}} \theta_f \phi_f(S_t, x_t) \right). \tag{48}$$

35

Assume we use ADP-based algorithms to determine an estimate of θ. Now, using this value of θ as an initial estimate, perform policy search by solving

$$\max_{\theta} \sum_{t=0}^{T} C(S_t, X^{VFA}(S_t|\theta)).$$

A nice illustration of this strategy is given in Maxwell et al. (2013). It is quite likely that performing this additional tuning (which can be expensive) will further improve the results. After performing parameter tuning, we can no longer view the linear term as an approximation of the value of being in a state. After tuning the policy, this is a form of CFA with cost function correction term.

- PFA with anything - A policy function approximation is any mapping of state to action without solving an optimization problem. PFAs are the simplest and easiest to control, but they cannot solve complex problems. The remaining three classes of policies are all cost-based, which allows them to be used for much more complex problems (including problems where x_t is a high-dimensional vector). However, cost-based policies are harder to control.

  It is possible to create a hybrid of a PFA and any cost-based policy. Assume we are using a VFA-based policy $X^{VFA}(S_t|\theta^{VFA})$ (this could also be a direct lookahead or parametric CFA), which we would write as we did in equation (48), where we let θ^{VFA} be the coefficients in the value function approximation. Now assume we are given some parametric function (a PFA) that we represent using $X^{PFA}(S_t|\theta^{PFA})$. We can write a hybrid policy using parameter vector $\theta = (\theta^{VFA}, \theta^{PFA}, \theta^{PFA-VFA})$

$$X^{VFA-PFA}(S_t|\theta) = \arg\max_{x}\left(C(S_t, x) + \sum_{f\in\mathcal{F}} \theta_f \phi_f(S_t, x_t) + \theta^{PFA-VFA}\|x - X^{PFA}(S_t|\theta^{PFA})\|\right).$$
(49)

  where $\theta^{PFA-VFA}$ handles the scaling between the norm of the difference between x and the decision suggested by $X^{PFA}(S_t|\theta^{PFA})$ and the rest of the cost-based objective function.

These hybrid policies help to emphasize the reason why we need to state the objective (as we did in equations (19) - (23)) in terms of optimizing over *policies*.

6.4 Remarks

The academic literature is heavily biased toward the lookahead classes (VFAs and DLAs). These offer optimal policies, but computable optimal policies are limited to a very small class of problems: the linear control policy for LQR problems in optimal control, and lookup table value functions for problems with small state and action spaces, and a computable one-step transition matrix.

Approximating the value function has extended this framework, but only to a degree. Approximate/adaptive dynamic programming and Q-learning is a powerful tool, but again, algorithms that have been demonstrated empirically to provide near-optimal policies are rare. Readers have to realize that just because an algorithm enjoys asymptotic optimality (or attractive regret bounds) does not mean that it is producing near-optimal solutions in practice.

It is our belief that the vast majority of real-world sequential decision problems are solved with policies from the policy search class (PFAs and CFAs). PFAs have received some attention from the research literature (neural networks, linear/affine control policies). CFAs, on the other hand, are widely used in practice, yet have received minimal attention in the academic literature.

We would argue that PFAs and CFAs should have a place alongside parametric models in machine learning. A limitation is that they require a human to specify the structure of the parameterization, but this is also a feature: it is possible for domain experts to use their knowledge of a problem to capture structure. Most important is that PFAs and CFAs tend to be much simpler than policies based on value functions and lookaheads. But, *the price of simplicity is tunable parameters*, and tuning is hard.

6.5 Stochastic control, reinforcement learning, and the four classes of policies

The fields of stochastic control and reinforcement learning both trace their origins to a particular model that leads to an optimal policy. Stochastic control with additive noise (see equation (3)) produced an optimal policy from the original deterministic model with a quadratic objective function, given by $u_t = K_t x_t$. Reinforcement learning owes its origins to the field of Markov decision processes, which also produces an optimal policy for discrete problems, where the one-step transition matrix can be computed (see both (11) and (12)).

What then happened to both fields is the realization that these optimal policies can only be used in practice for a fairly narrow range of problems. For this reason, both communities evolved other strategies which can be viewed as being drawn from each of the four classes of policies we have described.

Optimal control The following policies can be found in the optimal control literature:

> **Policy function approximations** These describe a wide range of simple rules used in every day problems. Some examples are:
> - Buy low, sell high - These are simple rules for buying or selling assets.
> - (s, S) inventory policies - A widely used policy for inventory control is to place an order when the inventory $R_t < s$, in which case we order $S - R_t$.
> - Linear control laws - Drawing on the optimal policy $u_t = K_t x_t$ for the LQR problems, the controls community branched into general linear "control laws" which we might

write as
$$U^\pi(x_t|\theta) = \sum_{f \in \mathcal{F}} \theta_f \phi_f(x_t),$$
where someone with domain knowledge needs to choose the features $\phi_f(x_t)$, $f \in \mathcal{F}$, after which the parameters θ need to be tuned.

Value function approximation The optimal control literature was using neural networks to approximate "cost-to-go" functions since the 1970's (Werbos 1974). This strategy has been pursued in the controls literature under names including heuristic/neuro/ approximate/adaptive dynamic programming. See Si et al. (2004) for a nice summary of this line of research.

Direct lookahead The controls community has referred to policies based on optimizing over a planning horizon as *model predictive control*. The most common strategy is to solve deterministic lookahead models. Most optimal control problems are deterministic, but using a deterministic approximation of the future is typical even when the underlying problem is stochastic (see Camacho & Bordons (2003) and J.A. Rossiter (2004) for thorough introductions).

Parameterized MPC Some authors in model predictive control have realized that you can obtain better results by introducing parameters as we did in our energy storage problem to handle the uncertainty in forecasts. This work has been done under the umbrella of "robust MPC" (see Kothare et al. (1996) and Rakovic et al. (2012)). In our framework, this would be a hybrid direct lookahead-cost function approximation.

Reinforcement learning - The following policies are all contained in Sutton & Barto (2018):

Policy function approximation A popular policy for discrete action spaces is to choose an action based on the Boltzmann distribution given by
$$p(a|\theta, s) = \frac{e^{\theta\bar{\mu}_a}}{1 + \sum_{a' \in \mathcal{A}_s} e^{\theta\bar{\mu}_{a'}}}$$
where $\bar{\mu}_a$ is the current estimate (contained in the state s) of the value of action a. The policy is parameterized by θ, which can be optimized using several methods, one of which is known as the "policy gradient method."

In addition to using the policy gradient method on a Boltzmann policy, a number of papers approximate the policy with a neural network. If θ is the weights of the neural network, then we have a high-dimensional parameter search problem that we can approach using stochastic gradient algorithms (see, e.g. Spall (2003)), although the problem is not easy; simulating policies is noisy, and the problem is not convex.

Cost function approximation Upper confidence bounding (for multiarmed bandit problems) is a classic CFA:
$$X^{CFA}(S^n|\theta) = \arg\max_a \left(\bar{\mu}_a^n + \theta\sqrt{\frac{\ln n}{N_a^n}} \right)$$

where N_a^n is the number of times we have tried action a after n iterations. UCB policies enjoy nice regret bounds (Bubeck & Cesa-Bianchi 2012), but it is still important to tune θ.

VFA-based policy This would be Q-learning, where the policy is given by

$$X^{VFA}(S^n) = \arg\max_a \bar{Q}^n(S^n, a)$$

Direct lookaheads Monte Carlo tree search is a classic direct lookahead. Since MCTS is a stochastic lookahead, it has a policy within the lookahead policy. This policy looks like

$$\tilde{X}_{tt'}^{\pi}(\tilde{S}_{tt'}|\theta^{UCT}) = \arg\max_{\tilde{x}_{tt'}} \left(\tilde{C}(\tilde{S}_{tt'}, \tilde{x}_{tt'}) + \tilde{V}_{tt'}^x(\tilde{S}_{tt'}^x) + \theta^{UCT}\sqrt{\frac{\log N_x^n}{N^n(\tilde{S}_{tt'}, \tilde{x}_{tt'})}} \right) \quad (50)$$

Note that this lookahead policy uses both a value function approximation as well as a bonus term from upper confidence bounding. This logic is known as "upper confidence bounding on trees," abbreviated UCT. Thus, this is a hybrid policy (CFA with VFA) within a stochastic lookahead.

So, we see that both the optimal control and reinforcement learning communities are actively using strategies drawn from all four classes of policies. The same evolution has happened in the simulation-optimization community and the multi-armed bandit community. In the case of multi-armed bandit problems, there are actually distinct communities pursuing the different classes of policies:

- PFAs - Random sampling of experiments would constitute a PFA (this is a default policy, often used implicitly in statistics).

- CFAs - Upper confidence bounding is a popular policy for bandit problems in computer science (Bubeck & Cesa-Bianchi 2012).

- VFAs - The applied probability community has long used a decomposition technique to produce a series of dynamic programs which can be solved (one "arm" at a time) to obtain Gittins indices (Gittins et al. 2011).

- DLAs - These include expected improvement, knowledge gradient and kriging developed in applied math, operations research and geosciences (see the tutorial in Powell & Frazier (2008) or the book Powell & Ryzhov (2012) for overviews).

We now return to our energy storage problem.

$$
X_t^{PFA}(S_t|\theta) = \begin{cases} x_t^{EL} = \min\{L_t, E_t\} \\ x_t^{RL} = \begin{cases} \min\{L_t - x_t^{EL}, \min\{R_t, \rho^{dch}\}\} & \text{if } P_t > \theta^H \\ 0 & \text{if } P_t \le \theta^H \end{cases} \\ x_t^{GL} = L_t - x_t^{EL} - x_t^{RL} \\ x_t^{ER} = \min\{E_t - x_t^{EL}, \rho^{ch}, R^{max} - R_t\} \\ x_t^{GR} = \begin{cases} \min\{\rho^{ch} - x_t^{ER}, R^{max} - R_t - x_t^{ER}\} & \text{if } P_t < \theta^L \\ 0 & \text{if } P_t \ge \theta^L \end{cases} \\ x_t^{RG} = \begin{cases} \min\{R_t - x_t^{RL}, \rho^{dch} - x^{RL}\} & \text{if } P_t > \theta^H \\ 0 & \text{if } P_t \le \theta^H \end{cases} \end{cases}
$$

Figure 6: Parameterized policy function approximation for energy system in figure 3, with tunable parameters highlighted.

7 Policies for energy storage

We can illustrate all four classes of policies using our energy storage problem. Note that fully developing any of these classes would require a serious effort, so these are going to be little more than brief illustrations.

Policy function approximation - As a simple illustration of a policy function approximation, consider a buy-low, sell-high policy for handling the charging and discharging of a battery connected to the grid. This policy would be written

$$
X^{PFA}(S_t|\theta) = \begin{cases} +1 & p_t < \theta^{charge} \\ 0 & \theta^{charge} \le p_t \le \theta^{discharge} \\ -1 & p_t > \theta^{discharge} \end{cases} \tag{51}
$$

Figure 6 shows an example of a parameterized PFA for the energy system in figure 3, where we have highlighted four tunable parameters. Designing these policies (especially the one in figure 6) is an art that requires an understanding of the structure of the problem. Tuning is an algorithmic exercise.

It is important to recognize that we have written our PFAs using parameters θ that do not depend on time t, yet it is clear that in such a time-dependent setting (due to time-of-day patterns and rolling wind forecasts), the policy should be time dependent. However, tuning a two (or four) dimensional parameter vector θ is much easier than tuning a time-dependent parameter vector $(\theta_\tau)_{\tau=1}^{24}$.

Cost function approximation We are going to present a hybrid cost function approximation with a direct lookahead below.

Value function approximation We apply the VFA-based policy in equation (48) with the policy

$$
X^{VFA}(S_t|\theta) = \arg\max_x \left(C(S_t, x) + \left(\theta_1 R_t + \theta_2 R_t^2 + \theta_3 (x_t^{EB} + x_t^{ED})^2 + \theta_4 (x_t^{ED} + x_t^{BD} + x_t^{GD})^2 \right) \right).
$$

There are a variety of strategies for fitting the coefficients θ that have been developed under headings of reinforcement learning (Sutton & Barto 2018), approximate dynamic programming (see e.g. Powell (2011)) and adaptive dynamic programming (Si et al. 2004). Jiang et al. (2014) describes an extensive series of tests using value function approximations, where we found that VFA-based policies only worked well when we could exploit structure (such as concavity).

Direct lookahead For time-varying problems with a rolling forecast, a natural choice is to do a deterministic lookahead. We do this by setting up a time-staged linear programming model to optimize all decisions over a planning horizon. This is a deterministic lookahead model, so we let the decisions in the lookahead model created at time t be represented by $\tilde{x}_{tt'}$, which produces

$$X^{DLA}(S_t) \;=\; \arg\max_{x_t,(\tilde{x}_{tt'},t'=t+1,\ldots,t+H)} \left(p_t(x_t^{GB} + x_t^{GD}) + \sum_{t'=t+1}^{t+H} \tilde{p}_{tt'}(\tilde{x}_{tt'}^{GB} + \tilde{x}_{tt'}^{GD}) \right) \quad (52)$$

subject to the following constraints. First, for time t we have:

$$x_t^{BD} - x_t^{GB} - x_t^{EB} \;\leq\; R_t, \quad (53)$$

$$\tilde{R}_{t,t+1} - (x_t^{GB} + x_t^{EB} - x_t^{BD}) \;=\; R_t, \quad (54)$$

$$x_t^{ED} + x_t^{BD} + x_t^{GD} \;=\; D_t, \quad (55)$$

$$x_t^{EB} + x_t^{ED} \;\leq\; E_t, \quad (56)$$

$$x_t^{GD}, x_t^{EB}, x_t^{ED}, x_t^{BD} \;\geq\; 0. \quad (57)$$

Then, for $t' = t+1, \ldots, t+H$ we have:

$$\tilde{x}_{tt'}^{BD} - \tilde{x}_{tt'}^{GB} - \tilde{x}_{tt'}^{EB} \;\leq\; \tilde{R}_{tt'}, \quad (58)$$

$$\tilde{R}_{t,t'+1} - (\tilde{x}_{tt'}^{GB} + \tilde{x}_{tt'}^{EB} - \tilde{x}_{tt'}^{BD}) \;=\; \tilde{R}_{tt'}, \quad (59)$$

$$\tilde{x}_{tt'}^{ED} + \tilde{x}_{tt'}^{BD} + \tilde{x}_{tt'}^{GD} \;=\; f_{tt'}^D, \quad (60)$$

$$\tilde{x}_{tt'}^{EB} + \tilde{x}_{tt'}^{ED} \;\leq\; f_{tt'}^E. \quad (61)$$

Hybrid DLA-CFA The policy defined by the lookahead model given by equations (52) - (61) does not make any provision for handling uncertainty. The most significant source of uncertainty is the forecast of wind, which is represented deterministically in equation (61). One idea is to parameterize this constraint by replacing it with

$$\tilde{x}_{tt'}^{EB} + \tilde{x}_{tt'}^{ED} \;\leq\; \theta_{t'-t} f_{tt'}^E. \quad (62)$$

Now we would write the policy as $X^{DLA-CFA}(S_t|\theta)$ where $\theta = (\theta_\tau)_{\tau=1}^H$ is a set of coefficients for a rolling set of forecasts over a horizon of length H. It is very important to note that θ is not time-dependent, which means that a policy that needs to behave differently at different times of day becomes a stationary policy, because the forecasts capture all the time dependent information, and the forecasts are captured in the state variable.

The policies in the policy search class, given by $X^{PFA}(S_t|\theta)$ in equation (51), and $X^{DLA-CFA}(S_t|\theta)$ using the parameterized constraint (62), both need to be tuned by solving

$$\max_{\theta} F^{\pi}(\theta) = \mathbb{E}_{S_0} \mathbb{E}_{W_1,...,W_T|S_0} \sum_{t=0}^{T} C(S_t, X^{\pi}(S_t|\theta)), \qquad (63)$$

where S_t is governed by the appropriate system model as illustrated in section 5, and associated information process.

It is unlikely that anyone would test all four classes of policies to see which is best. A notable exception is Powell & Meisel (2016) which showed that any of the four classes of policies (or a hybrid) can work best by carefully choosing the data. It is important to realize that the four classes of policies are meta-classes: simply choosing a class does not mean that your problem is solved. Each class is actually a path to an entire range of strategies.

8   Extension to multiagent systems

We can easily extend our framework to multiagent systems by using the framework to model the environment associated with each agent. Let $\mathcal{Q}$ be the set of agents and let $q \in \mathcal{Q}$ represent a specific agent. There are four types of agents:

- The ground truth agent - This agent cannot make any decisions, or perform any learning (that is, anything that implies intelligence). This is the agent that would know the truth about unknown parameters that we are trying to learn, or which performs the modeling of physical systems that are being observed by other agents. Controlling agents are, however, able to change the ground truth.

- Controlling agents - These are agents that make decisions that act on other agents, or the ground truth agent (acting as the environment). Controlling agents may communicate information to other controlling and/or learning agents.

- Learning agents - These agents do not make any decisions, but can observe and perform learning (about the ground truth and/or other controlling agents), and communicate beliefs to other agents.

- Combined controlling/learning agents - These agents perform learning through observations of the ground truth or other agents, as well as making decisions that act on the ground truth or other agents.

Now take every variable in our framework and introduce the index q. So, S_{tq} would be the state

42

of the system for agent q at time t, which includes:

$$R_{tq} = \text{The state of resources controlled by agent } q \text{ at time } t.$$

$$I_{tq} = \text{Any other information known to agent } q \text{ at time } t.$$

$$B_{tq} = \text{The beliefs of agent } q \text{ about anything known to any other agent (and therefore not}$$
known to agent q). This covers parameters in the ground truth, anything known by any other agent (for example, the resources that an agent q' might be controlling), and finally, beliefs about how other agents make decisions.

Belief states are the richest and most challenging dimension of multiagent systems, especially when there is more than one controlling agent, as would occur in competitive games.

Decisions for agent q are represented by x_{tq}. In addition to decisions that act on the environment, decisions in multiagent systems can include both information collection and communication to other controlling and/or learning agents. Exogenous information arriving to agent q would be given by W_{tq}. The exogenous information may be observations of a ground truth, or decisions made by other agents. The transition function gives the equations for updating S_{tq} from decision x_{tq} and exogenous information $W_{t+1,q}$. The objective function captures the performance metrics for agent q.

The design of policies are drawn from the same four classes that we have described above.

One of the controlling agents may play the role of a central agent, but in this framework, a "central agent" is simply another agent who makes decisions that are communicated to "field agents" who then use these decisions in their planning.

There is a tendency in the literature on multiagent systems to work with a "system state" $S_t = (S_{tq})_{q \in \mathcal{Q}}$. We would take the position that this is meaningless, since no agent ever sees all this information. We would approach the modeling of each agent as its own system, with the understanding that a challenge of any intelligent agent is to develop models that help the agent to forecast the exogenous information process W_{tq}. Of course, this depends on the policy being used by the agent.

A careful treatment of the rich problem of multiagent systems is beyond the scope of this chapter. However, we feel that the modeling of multiagent systems using this approach, drawing on the four classes of policies, opens up new strategies for the modeling and control of distributed systems.

9   Observations

We are not the first to bridge optimal control with reinforcement learning. Recht (2019) highlights recent successes of reinforcement learning in AlphaGo (Fu 2017), and suggests that these methods should be adapted to control problems. We would argue that both fields have explored methods

that could benefit the other, although we note that the controls community introduced the idea of direct lookaheads (model predictive control) in the 1950's (a form of direct lookahead), affine policies in the 1960's (a form of policy function approximation), and value function approximations in the 1970's. Both communities have addressed "model-free" and "model-based" settings, and both have explored methods from all four classes of policies (although neither have investigated parametric CFAs in depth). We think the biggest difference between optimal control and reinforcement learning is the core motivating applications of each field: optimal control grew originally out of continuous problems of controlling physical devices (aircraft, rockets, robots) while reinforcement learning grew out of problems with discrete action spaces.

We close with the following observations:

1) The fields of stochastic control and reinforcement learning address sequential decision problems. We feel that this perspective identifies a range of problems that is much wider than the problems that have been traditionally associated with these communities.

2) There seems to be considerable confusion about the meaning of "reinforcement learning." In the 1990's and early 2000's, reinforcement learning was a method, called Q-learning. Today, it covers the entire range of methods described in section 6. If we accept that the four classes of policies are universal, then it means that reinforcement learning covers *any* policy for a sequential decision problem (the same is true of stochastic control).

3) Parametric CFAs are often derided by the stochastic optimization community ("heuristics," "deterministic approximations" are often heard), yet are widely used in practice. Properly designed parametric policies, however, can be surprisingly effective for two reasons: a) the parameterization can capture domain knowledge that is completely ignored with policies based on lookaheads (VFAs or stochastic DLAs), and b) tuning parameters in a realistic, stochastic base model which avoids the different approximations needed in stochastic lookaheads, can capture complex behaviors that would be overlooked using a simplified stochastic lookahead model.

4) Stochastic optimal control also addresses sequential decision problems, using a more flexible and scalable modeling framework. We have argued that the control community is also using instances of all four classes of policies. So, what is the difference between stochastic optimal control and reinforcement learning?

5) Our universal framework, which draws heavily on the language used by the stochastic control community, broadens the scope of both of these fields to any sequential decision problem, which we would argue is broader than the problem classes considered by either community. Further, we have drawn on the four classes of policies identified in Powell (2019) which encompasses all the strategies already being explored by both communities. Since our classes are general (they are better described as meta-classes), they help guide the design of new strategies, including hybrids.

References

Astrom, K. J. (1970), *Introduction to Stochastic Control Theory*, Dover Publications, Mineola, NY.

Bellman, R. E. (1957), *Dynamic Programming*, Princeton University Press, Princeton, N.J.

Bellman, R. E., Glicksberg, I. & Gross, O. (1955), 'On the Optimal Inventory Equation', *Management Science* **1**, 83–104.

Bertsekas, D. P. & Shreve, S. E. (1978), *Stochastic Optimal Control: The discrete time case*, Vol. 0, Academic Press, New York.

Bertsekas, D. P. & Tsitsiklis, J. N. (1996), *Neuro-Dynamic Programming*, Athena Scientific, Belmont, MA.

Bertsekas, D. P., Tsitsiklis, J. N. & Wu, C. (1997), 'Rollout Algorithms for Combinatorial Optimization', *Journal of Heuristics* **3**(3), 245–262.

Bouzaiene-Ayari, B., Cheng, C., Das, S., Fiorillo, R. & Powell, W. B. (2016), 'From single commodity to multiattribute models for locomotive optimization: A comparison of optimal integer programming and approximate dynamic programming', *Transportation Science* **50**(2), 1–24.

Browne, C. B., Powley, E. J., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S. & Colton, S. (2012), 'A Survey of Monte Carlo Tree Search Methods', *IEEE Transactions on Computational Intelligence and AI in Games* **4**(1), 1–49.

Bubeck, S. & Cesa-Bianchi, N. (2012), 'Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems', *Foundations and Trends® in Machine Learning* **5**(1), 1–122.

Camacho, E. & Bordons, C. (2003), *Model Predictive Control*, Springer, London.

Chang, H. S., Fu, M. C., Hu, J. & Marcus, S. I. (2005), 'An Adaptive Sampling Algorithm for Solving Markov Decision Processes', *Operations Research* **53**(1), 126–139.

Cinlar, E. (2011), *Probability and Stochastics*, Springer, New York.

Coulom, R. (2007), Efficient selectivity and backup operators in Monte-Carlo tree search, *in* 'Computers and games', Springer Berlin, Heidelberg, pp. 72–83.

DeGroot, M. H. (1970), *Optimal Statistical Decisions*, John Wiley and Sons.

Fu, M. C. (2017), Markov Decision Processes, AlphaGo, and Monte Carlo Tree Search: Back to the Future, *in* 'TutORials in Operations Research', pp. 68–88.

Gittins, J. & Jones, D. (1974), A dynamic allocation index for the sequential design of experiments, *in* J. Gani, ed., 'Progress in statistics', North Holland, Amsterdam, pp. 241—-266.

Gittins, J., Glazebrook, K. D. & Weber, R. R. (2011), *Multi-Armed Bandit Allocation Indices*, John Wiley & Sons, New York.

J.A. Rossiter (2004), *Model-Based Predictive Control*, CRC Press, Boca Raton, FL.

Jiang, D. R., Pham, T. V., Powell, W. B., Salas, D. F. & Scott, W. R. (2014), A comparison of approximate dynamic programming techniques on benchmark energy storage problems: Does anything work?, *in* 'IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning', IEEE, Orlando, FL, pp. 1–8.

Kirk, D. E. (2004), *Optimal Control Theory: An introduction*, Dover, New York.

Kothare, M. V., Balakrishnan, V. & Morari, M. (1996), 'Robust constrained model predictive control using linear matrix inequalities', *Automatica* **32**(10), 1361–1379.

Kushner, H. (1971), *Introduction to stochastic control*, Holt, Rinehart and Winston, New York.

Lazaric, A. (2019), Introduction to Reinforcement Learning, *in* 'https://www.irit.fr/cimi-machine-learning/sites/irit.fr.CMS-DRUPAL7.cimi_ml/files/pictures/lecture1_bandits.pdf'.

Lewis, F. L. & Vrabie, D. (2009), 'Reinforcement Learning and Adaptive Dynamic Programming for Feedback Control', *IEEE Circuits And Systems Magazine* **9**(3), 32–50.

Lewis, F. L., Vrabie, D. & Syrmos, V. L. (2012), *Optimal Control*, 3rd edn, John Wiley & Sons, Hoboken, NJ.

Maxwell, M. S., Henderson, S. G. & Topaloglu, H. (2013), 'Tuning approximate dynamic programming policies for ambulance redeployment via direct search', *Stochastic Systems* **3**(2), 322–361.

Murray, J. J., Member, S., Cox, C. J., Lendaris, G. G., Fellow, L. & Saeks, R. (2002), 'Adaptive Dynamic Programming', *IEEE Transactions on Systems, Man, and Cybernetics - Part C Applications and Reviews* **32**(2), 140–153.

Nisio, M. (2014), *Stochastic Control Theory: Dynamic Programming Principle*, Springer, New York.

Powell, W. B. (2011), *Approximate Dynamic Programming: Solving the curses of dimensionality*, 2 edn, John Wiley & Sons, Hoboken, NJ.

Powell, W. B. (2014), 'Clearing the Jungle of Stochastic Optimization', *Bridging Data and Decisions* (January 2015), 109–137.

Powell, W. B. (2019), 'A unified framework for stochastic optimization', *European Journal of Operational Research* **275**(3), 795–821.

Powell, W. B. (2020), *Reinforcement Learning and Stochastic Optimization: A unified framework for sequential decisions*, Princeton NJ.

Powell, W. B. & Frazier, P. I. (2008), 'Optimal Learning', *TutORials in Operations Research* pp. 213–246.

Powell, W. B. & Meisel, S. (2016), 'Tutorial on Stochastic Optimization in Energy - Part II: An Energy Storage Illustration', *IEEE Transactions on Power Systems*.

Powell, W. B. & Ryzhov, I. O. (2012), *Optimal Learning*.

Puterman, M. L. (2005), *Markov Decision Processes*, 2nd edn, John Wiley and Sons, Hoboken, NJ.

Rakovic, S. V., Kouvaritakis, B., Cannon, M., Panos, C. & Findeisen, R. (2012), 'Parameterized tube model predictive control', *IEEE Transactions on Automatic Control* **57**(11), 2746–2761.

Recht, B. (2019), 'A Tour of Reinforcement Learning: The View from Continuous Control', *Annual Review of Control, Robotics, and Autonomous Systems* **2**(1), 253–279.

Sethi, S. P. (2019), *Optimal Control Theory: Applications to Management Science and Economics*, 3 edn, Springer-Verlag, Boston.

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

Simão, H., Day, J., George, A. P., Gifford, T., Nienow, J. & Powell, W. B. (2009), 'An approximate dynamic programming algorithm for large-scale fleet management: A case application', *Transportation Science*.

Sontag, E. (1998), 'Mathematical Control Theory, 2nd ed.', *Springer* pp. 1–544.

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

Stengel, R. F. (1986), *Stochastic optimal control: theory and application*, John Wiley & Sons, Hoboken, NJ.

Stengel, R. F. (1994), *Optimal Control and Estimation*, Dover Publications, New York.

Sutton, R. S. & Barto, A. G. (1998), *Reinforcement Learning: An Introduction*, MIT, Cambridge, MA.

Sutton, R. S. & Barto, A. G. (2018), *Reinforcement Learning: An Introduction*, 2nd edn, MIT Press, Cambridge, MA.

Vrabie, D. & Lewis, F. (2009), 'Neural network approach to continuous-time direct adaptive optimal control for partially unknown nonlinear systems', *Neural Networks* **22**(3), 237–246.

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

Yong, J. & Zhou, X. Y. (1999), *Stochastic Controls: Hamiltonian Systems and HJB Equations*, Springer, New York.