TUTORIALS IN
OPERATIONS RESEARCH
INFORMS 2016

 | ISBN 978-0-9843378-9-7

A Unified Framework for Optimization under Uncertainty
TutORials in Operations Research

Warren B. Powell
Department of Operations Research and Financial Engineering, Princeton University,
powell@princeton.edu

Abstract Stochastic optimization, also known as optimization under uncertainty, is studied by over a dozen communities, often (but not always) with different notational systems and styles, typically motivated by different problem classes (or sometimes different research questions) which often lead to different algorithmic strategies. This resulting "jungle of stochastic optimization" has produced a highly fragmented set of research communities which complicates the sharing of ideas. This tutorial unifies the modeling of a wide range of problems, from dynamic programming to stochastic programming to multiarmed bandit problems to optimal control, in a common mathematical framework that is centered on the search for policies. We then identify two fundamental strategies for finding effective policies, which leads to four fundamental classes of policies which span every field of research in stochastic optimization.

Keywords Stochastic optimization, stochastic control, dynamic programming, stochastic programming, multiarmed bandit problems, ranking and selection, simulation optimization, approximate dynamic programming, reinforcement learning, model predictive control, sequential learning

Contents

1. Introduction

Deterministic optimization, which comes in many forms such as linear/nonlinear/integer programming (to name a few), has long enjoyed a common mathematical framework. For example, researchers and academics over the entire world will write a linear program in the form

$$\min_x cx, \tag{1}$$

subject to

$$Ax = b, \tag{2}$$
$$x \leq u, \tag{3}$$
$$x \geq 0, \tag{4}$$

where x is a vector (possibly integer), c is a vector of cost coefficients, and A and b are suitably dimensioned matrices and vectors. There are various transformations to handle integrality or nonlinearities that are easily understood.

The same cannot be said of stochastic optimization, which is increasingly becoming known as optimization under uncertainty. Stochastic optimization has a long history of being highly fragmented, with names that include

- Decision trees
- Optimal control, including
  - Stochastic control
  - Model predictive control
- Stochastic search
- Optimal stopping
- Stochastic programming
- Dynamic programming, including
  - Approximate/adaptive dynamic programming
  - Reinforcement learning
- Simulation optimization
- Multiarmed bandit problems
- Online optimization
- Robust optimization
- Statistical learning

These communities are characterized by diverse terminologies and notational systems, often reflecting a history where the need to solve stochastic optimization problems evolved from a wide range of different application areas. Each of these communities start from well-defined canonical problems or solution approaches, but there has been a steady process of field creep as researchers within a community seek out new problems, sometimes adopting (and reinventing) methodologies that have been explored in other communities (but often with a fresh perspective).

Hidden in this crowd of research communities are methodologies that can be used to solve problems in other communities. A goal of this tutorial is to expose the universe of problems that arise in stochastic optimization, to bring them under a single, unified umbrella comparable to that enjoyed in deterministic optimization.

The tutorial begins in section 2 with a summary of the dimensions of stochastic optimization problems, which span static through fully sequential problems. Section 3 describes different modeling styles, and then chooses a particular modeling system for our framework. Section 4 describes a series of canonical problems to help provide a base of reference for readers from different communities, and to illustrate the breadth of our framework. Section 5

provides a brief introduction to belief models which play a central role in sequential learning policies.

Then, section 6 provides a tour from the canonical static stochastic search problem to fully sequential problems (dynamic programs), and back. This section presents a series of observations that identify how this diverse set of problems can be modeled within a single framework. Section 7 takes a brief detour to build bridges to two particular problem settings: learning with a dynamic, exogenous state (sometimes referred to as "contextual bandits") and the entire field of statistical learning, opening an entirely new path for unification. Having shown that optimizing over policies is the central modeling device that unifies these problems, section 8 provides a roadmap by identifying two fundamental strategies for designing policies.

2. Dimensions of a stochastic optimization problem

The field of math programming has benefitted tremendously from a common canonical framework consisting of a decision variable, constraints, and an objective function. This vocabulary is spoken the world over, and has helped serve as a basis for highly successful commercial packages. Stochastic optimization has not enjoyed this common framework.

Below we describe the dimensions of virtually any stochastic optimization problem, followed by a rundown of problem classes based on the staging of decisions and information.

2.1. Dimensions of a problem

We begin by identifying five dimensions of any stochastic optimization problem:

- State variable - The state variable is the minimally dimensioned function of history that captures all the information we need to model a system from some point in time onward. The elements of a state variable can be divided into three classes:
  - Physical state - This might capture the amount of water in a reservoir, the location of a piece of equipment, or speed of an aircraft.
  - Informational state - This includes other information, known deterministically, that is not included in the physical state.
  - Knowledge (or belief) state - This captures the probability distributions that describe the uncertainty about unknown static parameters, or dynamically evolving (but unobservable) states.

  The difference between physical and informational state variables is not important; these are distinguished simply because there is a natural tendency to equate "state" with "physical state." The knowledge state captures any information that is only known probabilistically (technically this includes the physical or information states, which is simply information known deterministically).
- Decisions/actions/controls - These can come in a variety of forms:
  - Binary
  - Discrete set (categorical)
  - Continuous (scalar or vector)
  - Vector integer (or mixed continuous and integer)
  - Subset selection
  - Vector categorical (similar to discrete set, but very high-dimensional)
- Exogenous information - This describes new information that arrives over time from an exogenous (uncontrollable) source, which are uncertain to the system before the information arrives. There are a number of different uncertainty mechanisms such as observational uncertainty, prognostic (forecasting) uncertainty, model uncertainty and implementation uncertainty (to name a few), which can be described using a variety of distributions: binomial, thin-tailed, heavy-tailed, bursts, spikes, and rare events.

- Transition function - These are functions which describe how the state of the system evolves over time due to endogenous decisions and exogenous information. This may be known or unknown, and may exhibit a variety of mathematical structures (e.g. linear vs. nonlinear). The transition function describes the evolution of all the state variables, including physical and informational state variables, as well as the state of knowledge.
- Objective function - We always assume the presence of typically one metric (some applications have more) that can be used to evaluate the quality of decisions. Important characteristics of objective functions include:
  · Differentiable vs. nondifferentiable
  · Structure (convex/nonconvex, monotone, low-rank, linear)
  · Expensive vs. inexpensive function evaluations
  · Final or cumulative costs (or regret)
  · Uncertainty operators, including expectations, conditional value at risk (CVaR), quantiles, and robust objectives (min max).

Remark 1. Some stochastic optimization problems require finding a single decision variable/vector that works well (according to some metric) across many different outcomes. More often, we are looking for a function that we refer to as a *policy* (also referred to as a decision function or control law) which is a mapping from state to a feasible decision (or action or control). Finding effective (ideally optimal) policies is the ultimate goal, but to do this, we have to start from a proper model. That is the central goal of this article.

2.2. Staging of information and decisions

It is useful to distinguish problem classes in terms of the staging of information and decisions. Below we list major problem classes, and describe each in terms of the sequencing of decisions and information.

- Offline stochastic search - Decision-information
- Online learning - Decision-information-decision-information ...
- Two-stage stochastic programming - Decision-information-decision
- Multistage stochastic programming - Decision-information-decision-information ...-decision-information
- Finite horizon Markov decision processes - Decision-information-decision-information ...-decision-information
- Infinite horizon Markov decision process - Decision-information-decision-information ...

All of these problems are assumed to be solved with an initial static state S_0 (which may include a probability distribution describing an unknown parameter), which is typically fixed. However, there is an entire class of problems where each time we perform a function evaluation, we are given a new state S_0. These problems have been described as "contextual bandits" in the machine learning community, optimization with an "observable state," and probably a few other terms. We can modify all of the problems above by appending initial "information" before solving the problem. We return to this important problem class in more depth in section 7.1.

3. Modeling sequential decision problems

If we are going to take advantage of the contributions of different fields, it is important to learn how to speak the languages of each communities. We start by reviewing some of the major notational systems used in stochastic optimization, followed by a presentation of the notational system that we are going to adopt.

3.1. Notational systems

To present the universe of problems in stochastic optimization it is useful to understand the different notational systems that have evolved to model these problems. Some of the most commonly used notation for each of the elements of a problem include:

- State variables - These are typically modeled as S_t (or s_t or s) in the dynamic programming literature, or x_t in the optimal control literature.
- Decisions - The most common standard notations for decisions are
  - a_t - Discrete actions.
  - u_t - Continuous controls, typically scalar, but often vector-valued with up to 10 or 20 dimensions.
  - x_t - Typically vectors, may be continuous, or discrete (binary or general). In operations research, it is not unusual to work with vectors with tens to hundreds of thousands of variables (dimensions), but even larger problems have been solved.
- Exogenous information - There is very little standard notation when it comes to modeling exogenous information (random variables). There are two fairly standard notational systems for sample realizations: s for "scenario" and ω for "sample path." While the language of scenarios and sample paths is sometimes used interchangeably, they actually have different meanings. In Markov decision processes, the random process is buried in the one-step transition matrix $p(s'|s,a)$ which gives the probability of transitioning to state s' when you are in state s and take action a. Notation for random variables for the new information arriving at time t includes ξ_t, w_t, ω_t, $\bar{\omega}_t$, and X_t.
- Transition functions - The control theory community uses the concept of a transition function more widely than any other community, where the standard notation is $x_{t+1} = f(x_t, u_t)$ (for a deterministic transition) or $x_{t+1} = f(x_t, u_t, w_t)$ for a stochastic transition (where x_t is the state variable, u_t is the control, and w_t is the "noise" which is random at time t). The operations research community will typically use systems of linear equations linking decisions across time periods such as

  $$A_t x_t + B_{t-1} x_{t-1} = b_t,$$

  where x_t is a decision variable. This style of writing equations follows the standard protocol of linear programming, where all decision variables are placed to the left of the equality sign; this style does not properly represent the dynamics, and does not even attempt to model a state variable.
- Objective function - There are a number of variables used to communicate costs, rewards, losses and utility functions. The objective function is often written simply as $F(x, W)$ where x is a decision variable and W is a random variable, implying that we are minimizing (or maximizing) $\mathbb{E}F(x, W)$. Linear costs are typically expressed as a coefficient c_t (we might write total costs at time t as $c_t x_t$, $c_t^T x_t$ or $\langle c_t, x_t \rangle$), while it is common in dynamic programming to write it as a general function of the state and action (as in $C(S_t, a_t)$). $g(\cdot)$ (for gain), $r(\cdot)$ (for reward), and $L(\cdot)$ (or $\ell(\cdot)$) (for losses) are all common notations in different communities. Instead of maximizing a reward or minimizing a cost, we can minimize regret or opportunity cost, which measures how well we do relative to the best possible.

Authors exercise considerable independence when choosing notation, and it is not uncommon for the style of a single author to evolve over time. However, the discussion above provides a high-level perspective of some of the most widely used notational systems.

In our presentation below, we will switch from using time $t = 0, \ldots, T$, which we index in the subscript, and iteration counters $n = 0, \ldots, N-1$, since each of these systems is best suited to certain settings. We start our iteration counter at $n = 0$ for consistency with how we index time (it also makes it easier to use notation such as θ^0 as our prior). There are

problems where we will iteratively simulate over a finite horizon, in which case we might let x_t^n be a decision we make at time t while following sample path ω^n. Below, we will switch from sampling at time t (using decision x_t) or iteration n (using decision x^n) reflecting the context of the problem (and the style familiar to the community that works on a problem).

3.2. A canonical notational system

Choosing a notational system requires navigating different communities. We have evolved the following notational system:

- State variable - S_t is the minimally dimensioned function of history that captures all the information needed to model the system from time t onward (when combined with the exogenous information process). The initial state S_0 captures all information (deterministic or distributional) that we are given as input data. We then limit S_t to include only information that is changing over time (we implicitly allow the system to use any deterministic, static data in S_0). We note that this definition (which is consistent with that used in the controls community) means that *all* properly modeled systems are Markovian.
- Decisions/actions/controls - We use a_t to refer to discrete actions, and x_t to represent decisions that may be vector-valued, as well as being continuous or discrete. We would reserve u_t for problems that are similar to engineering control problems, where u_t is continuous and low-dimensional. When dealing with sequential problems, we assume that we need to find a function, known as a *policy* (or *control law* in engineering), that maps states to decisions (or actions, or controls). If we are using a, u, or x for action, control or decision, we denote our policy using $A^\pi(S_t)$, $U^\pi(S_t)$ or $X^\pi(S_t)$, respectively. In this notation, "π" carries information about the structure of the function, along with any tunable parameters (which we tend to represent using θ). These are all stationary policies. If the policy is time dependent, then we might write $X_t^\pi(S_t)$, for example.
- Exogenous information - We let W_t be the information that first becomes known at time t (or between $t-1$ and t). Our use of a capital letter is consistent with the style of the probability community. It also avoids confusion with w_t used in the control community, where w_t is random at time t. When modeling real problems, we have to represent specific information processes such as prices, demands, and energy from wind. In this case, we put a "hat" on any variables that are determined exogenously. Thus, $\hat{p}_t$ might be the change in a price between $t-1$ and t; $\hat{D}_t$ might be the demand that was first revealed at time t. We would then write $W_t = (\hat{p}_t, \hat{D}_t)$.
  For those that like the formality, we can let $\omega \in \Omega$ be a sample realization of the sequence $W_1, \ldots, W_T$. Let $\mathcal{F}$ be the sigma-algebra that captures the set of events on Ω and let $\mathcal{P}$ be the probability measure on $(\Omega, \mathcal{F})$, giving us the standard probability space $(\Omega, \mathcal{F}, \mathcal{P})$. Probabilists like to define a set of sub-sigma-algebras (filtrations) $\mathcal{F}_t = \sigma(W_1, \ldots, W_t)$ generated by the information available up to time t. We note that our notation for time implies that any variable indexed by t is $\mathcal{F}_t$-measurable. However, we also note that a proper and precise model of a stochastic, dynamic system does not require an understanding of this mathematics (but it does require a precise understanding of a state variable).
- Transition function - The transition function (if known) is a set of equations that takes as input the state, decision/action/control, and (if stochastic), the exogenous information to give us the state at the next point in time. The control community (which introduced the concept) typically writes the transition function as $x_{t+1} = f(x_t, u_t)$ for deterministic problems, or $x_{t+1} = f(x_t, u_t, w_t)$ for stochastic problems (where w_t is random at time t). The transition function is known variously as the "plant model" (literally, the model of a physical production plant), "plant equation," "law of motion," "transfer function," "system dynamics," "system model," and "transition law," as well as "transition function." Since $f(\cdot)$ is used for so many purposes, we let

$$S_{t+1} = S^M(S_t, x_t, W_{t+1}) \tag{5}$$

be our transition function, which carries the mnemonic "state transition model." Applications where the transition function is unknown are often referred to as "model free"; an example might arise in the modeling of complex systems (climate, large factories) or the behavior of people. The term "model-based" would naturally mean that we know the transition function, although the reinforcement learning community often uses this term to mean that the transition *matrix* is known, which is typically written $p(s'|s,a)$ where $s = S_t$ is a discrete state, a is a discrete action, and $s' = S_{t+1}$ is a random variable given s and a. There are many problems where the transition function is known, but difficult or impossible to compute (typically because the state space is too large).

- Objective function - Below we use two notational systems for our contributions or costs. In certain settings, we use $F(x,W)$ as our contribution (or cost), reflecting the behavior that it depends only on our choice x and a random variable W. In other settings, we use $C(S_t,x_t)$ as our contribution, reflecting its dependence on the information in our state variable, along with a decision x_t. In some cases, the contribution depends on a random variable, and hence we will write $C(S_t,x_t,W_{t+1})$. There are many settings where it is more natural to write $C(S_t,x_t,S_{t+1})$; this convention is used where we can observe the state, but do not know the transition function. There are three styles for writing an objective function:

  Asymptotic form - We wish to solve

$$\max_x \mathbb{E}F(x,W). \tag{6}$$

  Here, we will design algorithms to search for the best value of x in the limit.

  Terminal reward - We may have a budget of N function evaluations, where we have to search to learn the best solution with some policy that we denote $x^{\pi,N}$, in which case we are looking to solve

$$\max_\pi \mathbb{E}\left\{F(X^{\pi,N},W)|S_0\right\}. \tag{7}$$

  Cumulative contribution - Problems that are most commonly associated with dynamic programming seek to maximize contributions over some horizon (possibly infinite). Using the contribution $C(S_t,x_t)$ and the setting of optimizing over time, this objective function would be written

$$\max_\pi \mathbb{E}\left\{\sum_{t=0}^{T} C(S_t,X_t^\pi(S_t))|S_0\right\}, \tag{8}$$

  where $S^{n+1} = S^M(S^n,X_t^\pi(S^n),W^{n+1})$.

  State-dependent information - The formulations above have been written under the assumption that the information process $W_1,\ldots,W_T$ is purely exogenous. There are problems where the information W_t may depend on a combination of the state S_t and/or the action x_t, which means that it depends on the policy. In this case, we would replace the $\mathbb{E}$ in (7) or (**??**)ith $\mathbb{E}^\pi$.

Remark 2. There is tremendous confusion about state variables across communities. The term "minimally dimensioned function of history" means the state variable S_t (or S^n) may include information that arrived before time t (or n). The idea that this is "history" is a complete misnomer, since information that arrives at time $t-1$ or $t-2$ is still known at time t (what matters is what is known, not when it became known). State variables may be complex; it is important to model first, and then deal with computational issues, since some policies handle complexity better than others. There is a tendency to associate multi-dimensional problems with the so-called "curse of dimensionality," but in fact the curse of dimensionality only arises when using lookup table representations. For example,

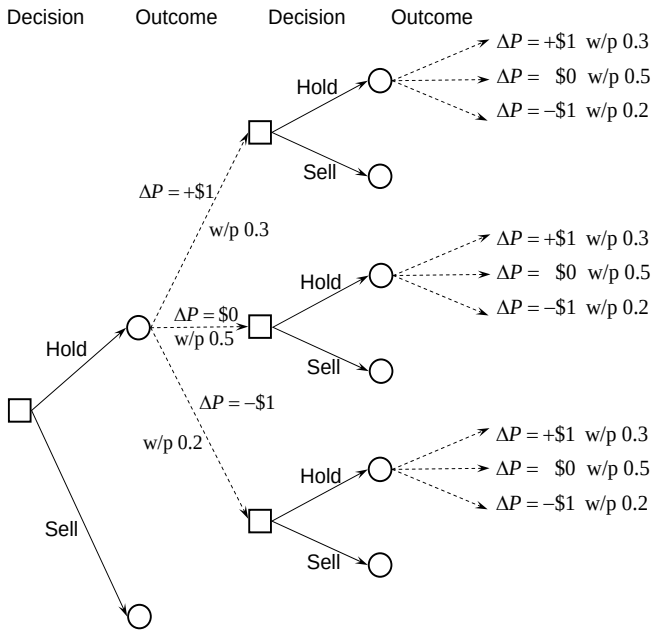

FIGURE 1. Illustration of a decision tree to determine if we should hold or sell a stock, where the stock might go up or down $1, or stay the same in each time period.

[56] describes a Markov decision process model of a large trucking company, where the state variable has 10^{20} *dimensions*. Please see [46][Section 3] for a careful discussion of state variables.

Remark 3. The controls community often refers to the transition function as "the model," but the term "model" is sometimes also used to include the exogenous information process, and the cost function. In operations research, the term "model" refers to the entire system: objective function, decision variables and constraints. Translated to our setting, "model" would refer to all five dimensions of a dynamic system, which is the approach we prefer.

4. Canonical problems

Each community in stochastic optimization seems to have a particular canonical problem which is used as an illustrative problem. These basic problems serve the valuable role of hinting at the problem class which motivates the solution approach.

4.1. Decision trees

Decision trees appear to have evolved first, and continue to this day to represent a powerful way of illustrating sequential decision problems, as well as a useful solution approach for many problems. Figure 1 illustrates a basic decision tree representing the problem of holding or selling a stock with a stochastically evolving price. This figure illustrates the basic elements of decision nodes (squares) and outcome nodes (circles). The decision tree is solved by stepping backward, computing the value of being at each node. The value of outcome nodes are computed by averaging over the outcomes (the cost/reward plus the downstream value), while the value of decision nodes is computed by taking the best of the decisions (cost/reward plus the downstream value).

Decision trees are easy to visualize, and are rarely expressed mathematically. Although decision trees date at least to the early 1800's, they represent fully sequential problems (decision-information-decision-information-...), which is the most difficult problem class. The difficulty with decision trees is that they grow exponentially in size, limiting their use to relatively short horizons, with small action sets and small (or sampled) outcomes.

4.2. Stochastic search

The prototypical stochastic search problem is written

$$\min_{x\in\mathcal{X}} \mathbb{E}F(x,W), \tag{9}$$

where x is a deterministic decision variable or vector, $\mathcal{X}$ is a feasible region, and W is a random variable. It is generally understood that the expectation cannot be computed exactly, which is the heart of why this problem has attracted so much interest. The basic stochastic search problem comes in a wide range of flavors, reflecting issues such as whether we have access to derivatives, the nature of x (scalar discrete, scalar continuous, vector, integer), the nature of W (Gaussian, heavy-tailed), and the time required to sample $F(x,W)$ (which may involve physical experiments). See [58] for an excellent introduction to this problem class.

This basic problem has been adopted by other communities. If $\mathcal{X}$ is a set of discrete choices, then (9) is known as the ranking and selection problem. This problem has been picked up by the simulation-optimization community, which has addressed the problem in terms of using discrete-event simulation to find the best out of a finite set of designs for a simulation (see [12]), although this field has, of late, expanded into a variety of other stochastic optimization problems [23].

A related family of problems replaces the expectation with a risk measure ρ:

$$\min_{x\in\mathcal{X}} \rho F(x,W), \tag{10}$$

There is a rich theory behind different risk measures, along with an accompanying set of algorithmic challenges. A careful discussion of these topics is beyond the scope of this tutorial, with the exception of robust optimization which we introduce next.

4.3. Robust optimization

Robust optimization evolved in engineering where the problem is to design a device (or structure) that works well under the *worst* possible outcome. This addresses the problem with (9) which may work well on average, but may encounter serious problems for certain outcomes. This can cause serious problems in engineering, where a "bad outcome" might represent a bridge failing or a transformer exploding.

Instead of taking an average via an expectation, robust optimization constructs what is known as an *uncertainty set* that we denote $\mathcal{W}$ (the standard notation is to let uncertainty be denoted by u, with the uncertainty set denoted by $\mathcal{U}$, but this notation conflicts with the notation used in control theory). The problem is canonically written as a cost minimization, given by

$$\min_{x\in\mathcal{X}} \max_{w\in\mathcal{W}} F(x,w). \tag{11}$$

This problem is easiest to solve if $\mathcal{W}$ is represented as a simple box. For example, if $w = (w_1,\ldots,w_K)$, then we might represent $\mathcal{W}$ as a simple box of constraints $w_k^{min} \le w_k \le w_k^{max}$ for $k = 1,\ldots,K$. While this is much easier to solve, the extreme points of the hypercube (e.g. the worst of all dimensions) are unlikely to actually happen, but these are then likely to hold the points $w \in \mathcal{W}$ that guide the design. For this reason, researchers represent $\mathcal{W}$ with an ellipsoid which represents the uncertainty set more realistically, but produces a much more difficult problem (see [4] for an excellent introduction to this field).

4.4. Multiarmed bandit problem

The original statement of the multiarmed bandit problem involves choosing from a finite set of alternatives $\mathcal{X} = \{1, 2, \ldots, M\}$ (known as "arms") where the goal is to find the alternative that produces the highest reward, given by an unknown contribution $F(x, W)$. This terminology is based on the rather odd story of learning the probability that a slot machine x(known as a "one-armed bandit") will return a reward, which requires playing the slot machine to collect information. Each alternative (slot machine) is referred to as an arm, and we are forced to spend money to learn, resulting in the use of cumulative rewards.

If we could compute $\mu_x = \mathbb{E}F(x, W)$, we would simply choose the alternative x with the largest value of μ_x. Since we do not know μ_x, we assume that we can make a series of observations x_n, $n = 0, \ldots, N-1$. Assume we use a sampling policy $X^\pi(S^n)$ where S^n captures our state of knowledge about the vector μ after n observations. We then observe $\hat{F}^{n+1} = F(x^n, W^{n+1})$ and update our state of knowledge using updating equations that we can represent simply using $S^{n+1} = S^M(S^n, x^n, W^{n+1})$. However, we have to incur whatever reward we receive while collecting the information.

The problem can be formulated as

$$\max_\pi \mathbb{E}\left\{\sum_{n=0}^{N-1} F(X^\pi(S^n), W^{n+1})|S_0\right\}. \tag{12}$$

Aside from the cosmetic difference of being formulated as a maximization problem (this is the classical form of a multiarmed bandit problem), this can be viewed as similar to (9) with the single difference that we use the cumulative rewards, since the basic problem assumes that we have to experience the rewards while we do our learning.

Although the roots of the multiarmed bandit problem started with a problem with discrete alternatives and where the objective function measures cumulative rewards, the research community focusing on "bandit problems" has expanded to include virtually any sequential learning problem where we are allowed to choose what to observe.

4.5. Optimal stopping

A classical problem in stochastic optimization is known as the optimal stopping problem (there are many references, but [14] is an early classic). Imagine that we have a stochastic process W_t which determines a reward $f(W_t)$ if we stop at time t. Let $\omega \in \Omega$ be a sample path of $W_1, \ldots, W_T$ (we are going to limit our discussion to finite horizon problems, which might represent a maturation date on a financial option). Let

$$X_t(\omega) = \begin{cases} 1 & \text{If we stop at time } t, \\ 0 & \text{Otherwise.} \end{cases}$$

Let τ be the time t when $X_t = 1$ (we assume that $X_t = 0$ for $t > \tau$). This notation creates a problem, because ω specifies the entire sample path, which seems to suggest that we are allowed to look into the future before making our decision at time t (don't laugh - this mistake is not just easy to make, it is actually a fairly standard approximation in the field of stochastic programming [6]).

To fix this, we require that the function X_t be constructed so that it depends only on the history $W_1, \ldots, W_t$. When this is the case τ is called a *stopping time*. The optimization problem can then be stated as

$$\max_\tau \mathbb{E}X_\tau f(W_\tau), \tag{13}$$

where we require τ to be a "stopping time." Mathematicians will often express this by requiring that τ (or equivalently, X_t) be an "$\mathcal{F}_t$-measurable function." We return to this terminology later.

4.6. Two-stage stochastic programming

Stochastic programming emerged in the 1950's as the first effort (initiated by George Dantzig) to introduce uncertainty into linear programs [17]. The prototypical stochastic program, widely known as a *two-stage stochastic program*, consists of the sequence of decision-information-decision. For example, imagine first deciding where to build a factory, then you see product demands, and then you ship from the factory to the customer (if we knew the demands in advance, we could do a better job of locating the factory).

The canonical two-stage stochastic programming problem (see [54]) would be written as

$$\min_{x_0} c_0 x_0 + \mathbb{E}Q(x_0, W_1) \tag{14}$$

subject to

$$A_0 x_0 = b_0, \tag{15}$$
$$x_0 \leq u_0, \tag{16}$$
$$x_0 \geq 0. \tag{17}$$

It is typically assumed that W_1 is a random variable defined over a discrete set of outcomes, or scenarios, that we represent using $\hat{\Omega}$ (this is typically a sample of a larger set of outcomes). The function $Q(x_0) = \mathbb{E}Q(x_0, W_1)$ is known as the recourse function and is given by

$$Q(x_0) = \sum_{\omega \in \hat{\Omega}} p(\omega) \min_{x_1(\omega), \omega \in \hat{\Omega}} c_1(\omega) x_1(\omega), \tag{18}$$

subject to

$$A_1(\omega) x_1(\omega) = b_0, \tag{19}$$
$$x_1(\omega) \leq u_1(\omega), \tag{20}$$
$$x_1(\omega) \geq 0. \tag{21}$$

Often, this is solved as a single optimization problem over $(x_0, x_1(\omega))_{\omega \in \hat{\Omega}}$ which can then be written

$$\min_{(x_0, x_1(\omega))_{\omega \in \hat{\Omega}}} \left(c_0 x_0 + \sum_{\omega \in \hat{\Omega}} p(\omega) \min_{x_1} c_1(\omega) x_1(\omega) \right), \tag{22}$$

subject to (15)-(17) and (19)-(21).

There are many applications where x_0 and x_1 are reasonably high-dimensional vectors (this is typical in logistics applications). Such problems can be quite large, and have attracted considerable attention, since the resulting deterministic problem has considerable structure (see [50] for one of the earliest and most widely cited examples).

4.7. Multi-stage stochastic programming

The natural extension of a two-stage stochastic (linear) program is the multi-stage version, which is written as

$$\min_{\substack{A_0 x_0 = b_0 \\ x_0 \geq 0}} \langle c_0, x_0 \rangle + \mathbb{E}_1 \left[\min_{\substack{B_0 x_0 + A_1 x_1 = b_1 \\ x_1 \geq 0}} \langle c_1, x_1 \rangle + \mathbb{E}_2 \left[\cdots + \mathbb{E}_T \left[\min_{\substack{B_{T-1} x_{T-1} + A_T x_T = b_T \\ x_T \geq 0}} \langle c_T, x_T \rangle \right] \cdots \right] \right]. \tag{23}$$

This very general model allows all information to be random and revealed over time. Thus, the exogenous information at time t is given by $W_t = (A_t, B_t, b_t, c_t), t = 1, \ldots, T$. The initial values A_0, B_0, b_0, c_0 are assumed to be deterministic components of the initial state of the system $S_0 = (A_0, B_0, b_0, c_0)$.

As written this is computationally intractable, but it states a problem that is the extension of decision trees to problems where a decision at time t is a vector x_t that has to satisfy a set of linear constraints.

4.8. Markov decision processes

Assume that we have a state variable S_t that takes on a (not too large) set of discrete values $\mathcal{S} = \{1, \ldots, S\}$, with a discrete set of actions $a \in \mathcal{A}$. Further assume we are given a one-step transition matrix $p(s'|s,a) = P[S_{t+1} = s'|S_t = s, a_t = a]$ for $s, s' \in \mathcal{S}$. If $C(s,a)$ is the contribution from being in state s and taking action a, we can characterize an optimal action using Bellman's equations which are written

$$V_t(S_t) = \max_{a_t \in \mathcal{A}} \big(C(S_t, a_t) + \sum_{s' \in \mathcal{S}} p(s'|S_t, a_t) V_{t+1}(s')\big). \tag{24}$$

Equation (24) can be solved by stepping backward in time, starting from a final time period where we might assume $V_T(S_T) = 0$. Once we have computed $V_t(S_t)$ for all times t and all states $S_t \in \mathcal{S}$. Given these value functions, we obtain a policy (decision rule) given by

$$A_t^*(S_t) = \arg\max_{a_t \in \mathcal{A}} \big(C(S_t, a_t) + \sum_{s' \in \mathcal{S}} p(s'|S_t, a_t) V_{t+1}(s')\big). \tag{25}$$

Bellman's equation is mathematically the same as rolling back the decision tree in figure 1, but there is one significant exception between the two models. Decision trees are typically drawn with a unique path from the root node to each other node in the decision tree. In a Markov decision process, the state S_t corresponds to a decision node, but there may be many ways of reaching a particular state S_t. Formulated as a decision tree, a Markov decision process would continue growing exponentially, while the formulation here restricts the set of decision nodes (states) to the set $\mathcal{S}$. This, in fact, was the central insight of Bellman in the development of dynamic programming.

Often overlooked is that equations (24) and (25) are the basis of the optimal solution of the optimization problem

$$\max_{\pi} \mathbb{E}\left\{\sum_{t=0}^{T} C(S_t, A_t^{\pi}(S_t))|S_0\right\}, \tag{26}$$

where $S_{t+1} = S^M(S_t, A_t^{\pi}(S_t), W_{t+1})$, where the expectation operator is over all possible sample paths for the exogenous information process $W_1, \ldots, W_T$. When faced with formulating a "dynamic program," most authors will write Bellman's equation in the form of equation (24). In fact, Bellman's equation (24) is an optimality condition, not an optimization problem. We address the problem of finding policies in section 8.

4.9. Optimal control

Deterministic optimal control is widely formulated as

$$J^* = \min_{u_t, 0 \le t \le T} \sum_{t=0}^{T} L_t(x_t, u_t), \tag{27}$$

where the state x_t evolves according to $x_{t+1} = f(x_t, u_t)$ where $f(x_t, u_t)$ is a known transition function capturing the dynamics of the system. Here, $L_t(x_t, u_t)$ is a "loss" to be minimized. The controls u_t may be subject to a set of constraints.

We note that while this problem is deterministic, it is formulated using a state variable x_t and a transition function $f(x_t, u_t)$, which are standard tools for stochastic problems. We note that time-dependent linear programs do not use either of these notational devices.

Now consider what happens when our problem is stochastic, which is modeled by introducing noise into the transition function using

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

where the noise w_t is random at time t. The objective function (27) might now be written

$$J^* = \min_{u_t, 0 \le t \le T} \mathbb{E} \sum_{t=0}^{T} L_t(x_t, u_t). \tag{28}$$

We have introduced an expectation, but it appears that we are still optimizing over a deterministic vector u_t, $t = 0, \ldots, T$. However, u_t is now a random variable since it depends on the evolution of the system. If ω corresponds to a sample path of $w_0, w_1, \ldots, w_T$, then we can write $u_t(\omega)$ to reflect the dependence of the control on the sample path. The problem is that writing $u_t(\omega)$ as a function of ω could be interpreted to mean that u_t has been chosen given all the information that has yet to come in the future. Mathematicians fix this problem by following the objective function (28) with a statement such as "where u_t is $\mathcal{F}_t$-measurable" or, equivalently, "where u_t is an admissible policy." Both statements mean simply that $u_t(\omega)$ is only allowed to reflect information that would be available by time t. We refer to this language as "MCCM" (mathematically correct, computationally meaningless) since it does not provide a path to computation.

A mathematically equivalent way of writing this problem is to write the control $u_t = U_t^\pi(x_t)$ as dependent only on the state x_t at time t, which means that the policy (control) is only a function of the past by construction. Using this notation, we assume that the index π carries information on the type of function $U_t^\pi(x_t)$. We can now write the objective function as

$$J^* = \min_{\pi} \mathbb{E} \sum_{t=0}^{T} L_t(x_t, U_t^\pi(x_t)). \tag{29}$$

We address the problem of searching over policies in section 8. There is a special case of optimal control problem known as linear, quadratic regulation where the loss function has the form

$$L_t(x_t, u_t) = (x_t)^T Q_t x_t + (u_t)^T R_t u_t.$$

For unconstrained problems, it is possible to show that an optimal policy has the form

$$u_t^* = K_t x_t, \tag{30}$$

where K_t is a suitably configured matrix that is a function of the matrices Q_t and R_t (see [5] or [35]). Policies that are linear in the state variable (such as (30)) are known in dynamic programming as "affine controllers" (or "affine policies"). They can also be written in the form

$$U^\pi(x_t|\theta) = \sum_{f \in \mathcal{F}} \theta_f \phi_f(x_t), \tag{31}$$

where $(\phi_f(x_t))_{f \in \mathcal{F}}$ is a set of suitably chosen features. In most applications, policies of the form (31) are not optimal; rather, we pose them as approximations and then search for the values of θ that perform the best. We revisit this idea in section 8.

5. Belief models

Many stochastic optimization problems involve sequential learning. These can be divided into two groups:

- Pure learning problems, where the set of feasible decisions at each iteration (or point in time) is independent of past decisions.

- Learning with a physical state - In these problems, decisions are constrained by past decisions, as might occur when moving over a graph or managing resources.

We let K^n (or K_t) represent our state of knowledge, but in this section, we are just going to use S^n.

In any learning problem, the state of knowledge is stored in a belief model, which may be frequentist or Bayesian. These models can be represented using one of several basic strategies:

- Lookup table belief models - Here we have a distribution of belief about $\mu_x = \mathbb{E}F(x,W)$ for each discrete alternative x, which can be represented using a Bayesian or frequentist belief model. Assuming our belief about μ_x is normally distributed, we can use two possible representations:
  - Independent beliefs - We let $\mu_x \sim N(\theta_x^n, \beta_x^n)$ where θ_x^n is our estimate after n observations of the mean of the random variable μ_x, while $\beta_x^n = 1/\sigma_x^{2,n}$ is the precision.
  - Correlated beliefs - Here we let $\Sigma_{xx'}^n = Cov^n(\mu_x, \mu_{x'})$ after n measurements. If we assume that the means μ_x are normally distributed, we would write $\mu \sim MVN(\theta^n, \Sigma^n)$.
- Parametric belief models - Again we have two choices:
  - Linear belief models - This may be written (after n measurements)

$$\mathbb{E}F(x,W) = \mu_x \approx \sum_{f\in\mathcal{F}} \theta_f^n \phi_f(x),$$

    where $(\phi_f(x))_{f\in\mathcal{F}}$ is a set of features extracted from x, and θ^n is the vector of coefficients at iteration n.
  - Nonlinear belief models - Here we refer to parametric models that are nonlinear in the parameter vector θ. If we assume that $\mathbb{E}F(x,W) = f(x|\theta)$ for some known function $f(x|\theta)$, we would let the random variable θ follow some distribution appropriate for the applications. This might still be multivariate normal, or we might assume that θ is one of a finite set $(\theta_1, \ldots, \theta_K)$ each with probability p_k^n, in which case we would write $S^n = (f(x|\theta), (\theta_k, p_k^n)_{k=1}^K)$. Nonlinear belief models also include a broad spectrum of powerful, general purpose models such as neural networks or support vector regression/machines.
- Nonparametric/locally parametric belief models - This covers a wide range of methods which produce local approximations.

We assume that any belief model is more than just a point estimate, but rather includes a distribution of possible models. Let S^n represent this distribution of beliefs (our state of knowledge) after n observations. If we are using a lookup table representation with correlated beliefs, we would write $S^n = (\theta^n, \Sigma^n)$ where θ_x^n is our estimate of the function $\mathbb{E}F(x,W)$, and Σ^n is the covariance matrix where $\Sigma_{x,y}^n$ is the covariance of our estimate of $\mathbb{E}F(x,W)$ and $\mathbb{E}F(y,W)$. A point estimate of our approximation of $F(x) = \mathbb{E}F(x,W)$ would be

$$\bar{F}^n(x) = \theta_x^n.$$

We might approximate $F(x)$ using a linear model such as

$$\bar{F}^n(x|\theta) = \sum_{f\in\mathcal{F}} \theta_f^n \phi_f(x),$$

Finally, we might represent $F(x)$ using a parametric model $f(x|\theta)$ where θ can take on one of a finite set of possible values $(\theta_1, \ldots, \theta_K)$ where $p_k^n = P[\theta = \theta_k | \mathcal{F}^n]$ is the probability that the true parameter vector $\theta = \theta_k$. In this case, our belief model is given by the probability vector $p^n = (p_k^n)_{k=1}^K$. In this case, our belief state is $S^n = p^n$ and our approximation is given by

$$\bar{F}^n(x|\theta) = \sum_{k=1}^K p_k^n f(x|\theta_k).$$

Belief models can be virtually any statistical model (see [28] for an excellent overview of a wide range of statistical models). What distinguishes statistical estimation in a learning setting is that we need access to recursive updating equations. While it is beyond the scope of this document, simple updating equations exist for each of the models above (see [45][Chapter 7] for a discussion of these methods). For our purposes, we simply assume that the updating equations (for our knowledge state) are captured in our transition function $S^{n+1} = S^M(S^n, x^n, W^{n+1})$.

It is known (but not widely) that an optimal policy for learning problems can be characterized using Bellman's optimality equation (see e.g. [19])

$$V^n(S^n) = \min_x \big(C(S^n, x) + \mathbb{E}\{V^{n+1}(S^{n+1})|S^n\}\big), \tag{32}$$

where S^n is a knowledge state and $C(S^n, x)$ might be some form of statistical error, although it could be a performance metric that we are maximizing (as we do elsewhere in this article). The key idea is that the knowledge state is no different than a physical state in terms of modeling the problem. While equation (32) is typically intractable for a learning problem, there are special cases where it can be solved. For example, if our knowledge state $S^n = (f(x|\theta), (\theta_k, p_k^n)_{k=1}^K)$ consists of a discrete vector $(p_k^n)_{k=1}^K$ and if outcomes W^{n+1} are discrete (and finite, as would happen if the information was a random number of discrete arrivals), then p_k^n is limited to a set of finite outcomes, making it possible to solve (32) optimally (for small values of n).

6. From static optimization to sequential, and back

In this section we are going to show that static stochastic search, multiarmed bandit problems, and fully sequential dynamic programs (or stochastic control problems) can all be formulated in the same way.

6.1. Derivative-based stochastic search - asymptotic analysis

We return to our static stochastic search problem which we repeat here for convenience

$$\max_x \mathbb{E}F(x, W). \tag{33}$$

If $F(x, W)$ is differentiable, a well-known solution procedure is the stochastic gradient algorithm due to [49] which produces a sequence of iterates x^n according to

$$x^{n+1} = x^n + \alpha_n \nabla_x F(x^n, W^{n+1}). \tag{34}$$

Under some fairly simple conditions on the stepsize α_n, this algorithm ensures that $x^n \to x^*$ where x^* solves (33). This can be described as a static stochastic optimization problem which we solve using an asymptotically optimal algorithm. This algorithmic strategy, along with its asymptotic analysis, is formulated and solved using the same framework as a deterministic optimization problem. We are going to refer to this problem shortly with a different perspective.

The optimization problem in (33) is sometimes called offline learning, because we do not care about the value of the function at intermediate solutions. Instead, we only care about the cost of the final solution, something that we can also call the *terminal cost*. However, the stochastic gradient algorithm in (34) can also be viewed as an online algorithm, because it adapts to a stochastic process that we may be observing in the field (rather than being generated within the computer). While we believe that "online" could (and perhaps should) refer to situations where we have to live with the rewards as they arrive, this term is not uniformly used in this way in the literature. For this reason, we refer to iterative algorithms such as (34) as sequential.

By contrast, a static (or batch) solution approach might be to replace the expectation over Ω with a sample $\hat{\Omega} \in \Omega$, allowing us to compute an approximate solution using

$$\hat{x} = \arg\max_x \frac{1}{N} \sum_{\hat{\omega} \in \hat{\Omega}} F(x, W(\hat{\omega})). \tag{35}$$

Equation (35) is known as the *sample average approximation* (see [33], [54]). It replaces (33) with a (potentially large) deterministic optimization problem with nice asymptotic properties as the sample N grows. Of course, the simplest approach is to simply replace the random variable W with its expectation, giving

$$\bar{x} = \arg\max_x F(x, \mathbb{E}W), \tag{36}$$

but $\bar{x}$ is no longer reflecting the range of outcomes of W.

6.2. The effect of horizon on problem formulation

The formulation in (33) assumes that we can test different values of x, after which we can observe $F(x, W)$, where the only goal is to find a single, deterministic x that solves (33). We only care about the final solution, not the quality of different values of x that we test while finding the optimal solution.

We have seen that an asymptotic analysis of (33) using a stochastic gradient algorithm such as (34) approaches the problem just as we would any deterministic optimization problem. In practice, of course, we have to limit our evaluation of different algorithms based on how well they do within some specified budget. In this section, we are going to see what happens when we formally approach the problem within a fixed budget, known in some communities as *finite time analysis* (see [2] and [39] for examples).

We begin by assuming that $F(x, W)$ is a reward that we incur each time we test x. For example, we might have a newsvendor problem of the form

$$F(x, W) = p \min\{x, W\} - cx. \tag{37}$$

Here, W would be interpreted as a demand and x is the supply, where p is the price at which newspapers are sold, and c is the purchase cost. We can use our newsvendor problem for computing the stochastic gradients we need in our stochastic gradient algorithm (34).

In practice we have to tune the stepsize formula. While there are many rules we could use (see [24]), we will illustrate the key idea using a simple stepsize rule known as Kesten's rule given by

$$\alpha_n(\theta) = \frac{\theta}{\theta + N^n},$$

where θ is a tunable parameter and N^n counts the number of times that the objective function $F(x^n, W^{n+1})$ has declined (which means the stepsize remains constant while the function is improving).

Now, our stochastic gradient algorithm (34) becomes a policy $X^\pi(S^n)$ with state $S^n = (x^n, N^n)$, and where π captures the structure of the rule (e.g. the stepsize rule in (34)) and any tunable parameters (that is, θ). Let $x^{\pi,N}$ be the solution x^n for $n = N$, where we include π to indicate that our solution $x^{\pi,N}$ after N function evaluations depends on the policy π that we followed to get there. The problem of finding the best stepsize rule can be stated as

$$\max_\pi \mathbb{E} F(x^{\pi,N}, W), \tag{38}$$

which states that we are looking for the best terminal value within a budget of N function evaluations. Of course, we do not have to limit our search over policies to comparing stepsize

rules. There are different ways of computing the gradient, such as gradient averaging or the stochastic version of mirror descent ([42]).

There is a substantial literature on stochastic optimization algorithms which prove asymptotic convergence, and then examine rate of convergence empirically. Experimental testing of different algorithms is forced to work with fixed budgets, which means that researchers are looking for the best solution within some budget. We would argue that (38) is stating the aspirational goal of finding the *optimal algorithm* for maximizing $\mathbb{E}F(x,W)$ in N iterations.

It is easy to assume that the offline learning setting (optimizing terminal cost or reward) does not involve the tradeoff of exploration and exploitation, but this is not the case, since we have to do the best we can to learn the best choice within our budget of N function evaluations. This tradeoff is avoided only in the setting of an infinite horizon problem, since we no longer face a budget constraint.

6.3. Sequential learning - terminal reward

Now return to the case where $\mathcal{X}$ is a set of discrete alternatives, and where we want to maximize the terminal reward. This problem has been studied since the 1950's under the name of ranking and selection (see [19] for a nice review of this early literature). This literature has been extended under the umbrella of the simulation-optimization community [12] where the original problem was to use discrete-event simulation to find the best out of a finite set of designs, although lately the field has broadened to encompass a much wider range of problems (see the edited volume [23] for a nice summary of this expansion).

In contrast with our stochastic gradient-based approach where we depend on derivative information to guide our search, we have to introduce the concept of learning, which means we have to create a belief model about the function $\mathbb{E}F(x,W)$. The simplest belief model is a lookup table where we assume that $\mu_x = \mathbb{E}F(x,W)$ is a random variable where, after n observations, our belief about μ_x is that it is normally distributed with mean θ_x^n and variance $\sigma_x^{2,n}$ (this is a Bayesian interpretation, although we could easily shift to a frequentist one). We can assume that beliefs are independent, or we may capture covariances where $\Sigma^n_{x,x'} = Cov(\mu_x, \mu_{x'})$. In this case, we would say that the vector $\mu \sim MVN(\theta^n, \Sigma^n)$ after n observations. We then let $S^n = (\theta^n, \Sigma^n)$ be our belief state, with an assumption of normality (belief states must always carry an explicit model of the uncertainty). Other belief models can be any of the representations reviewed in section 5.

Let $\bar{F}^{\pi,n}(x|\theta)$ be our point estimate of the function $F(x) = \mathbb{E}F(x,W)$ after n iterations following learning policy π. If we are using our lookup table representation, then $\bar{F}^{\pi,n}(x|\theta) = \theta_x^n$ (for notational consistency, we continue to write $\bar{F}(x|\theta)$ as dependent on a parameter vector θ). If we approximate $F(x)$ with a linear model, then we would write

$$\bar{F}^{\pi,n}(x|\theta) = \theta_0^n + \theta_1^n \phi_1(x) + \theta_2^n \phi_2(x) + \dots.$$

Our belief model could also be a nonlinear parametric model, a neural network, or even a nonparametric model (see [28] for a nice overview of methods in statistical learning, and [15] and [29] for presentations in optimization settings).

We can now write our stochastic optimization problem as a sequential decision problem. Starting with some belief state S^0, we make a decision $x^0 = X^\pi(S^0)$ using some policy, and then observe the outcome of the experiment W^1 resulting from experiment x^0, which then leads us to belief state S_1, and so on. We can write the sequence of states, actions and information as

$$(S^0, x^0 = X^\pi(S^0), W^1, S^1, x^1 = X^\pi(S^1), W^2, \dots, S^N).$$

Assume that we use as our solution after N observations the one that appears to be best based on our approximation, given by

$$x^{\pi,N} = \arg\max_x \bar{F}^{\pi,N}(x|\theta^N).$$

Our objective function, then, is given by

$$\max_{\pi} \mathbb{E}\left\{\max_{x} F(x^{\pi,N}, W)|S_0\right\}, \tag{39}$$

where S_0 captures our prior distribution of belief about $F(x)$. Since we cannot compute $\mathbb{E}F(x,W)$ exactly, we may evaluate our results based on our approximation, giving us the objective

$$\max_{\pi} \mathbb{E}\{\max_{x} \bar{F}^{\pi,N}(x|\theta)|S_0\}. \tag{40}$$

It is possible to show that (39) and (40) are equivalent if, for example, the approximation $\bar{F}(x|\theta)$ spans $\mathbb{E}F(x,W)$ (that is, $\bar{F}(x|\theta) = \mathbb{E}F(x,W)$ for some θ).

There has evolved a fairly substantial literature for solving offline (terminal cost) stochastic optimization problems that can be viewed collectively as a search for the best algorithm (policy). For example, [42] compares stochastic gradient methods to sample average approximation. Other authors have investigated response surface methods [25], along with a host of methods based on lookup table representations (for problems where $\mathcal{X}$ is a set of discrete alternatives).

One policy is based on maximizing the value of information, which has been referred to as the *knowledge gradient* which is literally the marginal value of information ([22], [47]). This has been explored for offline (terminal cost) problems where the value of information is given by

$$\nu_x^{KG,n} = \mathbb{E}\{\max_{x'} \bar{F}^{n+1}(x'|\theta)|S^n, x^n = x\} - \max_{x'} \bar{F}^n(x'|\theta). \tag{41}$$

Here, $\bar{F}^{n+1}(x'|\theta)$ represents the updated state of knowledge about $F(x) = \mathbb{E}F(x,W)$ after running experiment $x = x^n$ given our current state of knowledge S^n which includes both point and distributional information about $\bar{F}^n(x'|\theta)$. The knowledge gradient policy for offline (terminal cost) problems is given by

$$X^{KG}(S^n) = \arg\max_{x} \nu_x^{KG,n}. \tag{42}$$

This policy is asymptotically optimal [22], but it also enjoys the property that it is optimal if $N = 1$ which explains its behavior of fast initial learning. The knowledge gradient is also able to handle a range of belief models, including correlated lookup table [21], linear belief model [41], nonparametric models ([38], [3]) and nonlinear models [13]. We return to this class of policies below for different problem settings.

So, we found in section 6.2 that when we transition from an asymptotic to a finite time analysis, our problem changes from optimizing over x to one of optimizing over policies π. Said differently, this formulation poses (we believe for the first time) the question of finding the *optimal algorithm*, expressed in the form of finding the best policy. This question is moot for infinite horizon problems, since trivial policies such as random search or round-robin algorithms are easily seen to be asymptotically optimal (but with very slow convergence rates). When we transition from derivative-based methods to derivative-free (as is necessary when $\mathcal{X}$ is discrete), we have to introduce the idea of learning a belief model.

There is one important feature of the sequential learning problem with a final reward objective that we need to keep in mind. We are trying to find the best policy to collect information to learn our function, but the ultimate goal is to find a single decision $x^{\pi,N}$ (rather than a policy). We still face the problem of searching over functions (the policies for collecting information). However, this is the only problem we consider where the end result is a single decision x rather than a function.

We next consider what happens when we accumulate rewards as we progress.

6.4. Sequential learning - Cumulative cost

Now consider the case of our newsvendor problem where we have to accumulate rewards as we proceed. The finite-budget version of this problem is given by

$$\max_{\pi} \mathbb{E}\left\{\sum_{n=0}^{N-1} F(X^{\pi}(S^n), W^{n+1})|S_0\right\}, \tag{43}$$

where $F(x,W)$ is our newsvendor problem given in (37). Unlike our terminal cost criterion, it makes sense to formulate a (discounted) infinite-horizon version of the problem, which we would write as

$$\max_{\pi} \mathbb{E}\left\{\sum_{n=0}^{\infty} \gamma^n F(X^{\pi}(S^n), W^{n+1})|S_0\right\}. \tag{44}$$

The infinite horizon, discounted reward problem (44) has long been studied under the name of the multiarmed bandit problem [26] which we first introduced in section 4.4.

Work on this problem class initially started in the applied probability community (see [62], [19] for early reviews), which first formulated the learning problem as a dynamic program using the belief state S^n as the state variable. The resulting dynamic program required handling a multidimensional state variable (two times the number of alternatives for the case of normally distributed beliefs) which was computationally intractable. The first breakthrough came with the introduction of Gittins indices ([27], [26]) which reduces the problem to a series of dynamic programs defined for each alternative (arm) individually. The Gittins index has the form

$$\nu_x^{Gittins,n} = \theta_x^n + \Gamma\left(\frac{\sigma_x^n}{\sigma^W}, \gamma\right)\sigma^W, \tag{45}$$

where σ^W is the standard deviation of the observation noise of an experiment. The term $\Gamma\left(\frac{\sigma_x^n}{\sigma^W}, \gamma\right)$ is called the "Gittins index" (computed for problems where rewards have mean 0 and variance 1), which requires solving a dynamic program for a standard problem. Although computing $\Gamma\left(\frac{\sigma_x^n}{\sigma^W}, \gamma\right)$ is not in itself easy, it was viewed as a computational breakthrough, and spawned a substantial literature on solving "bandit problems" by finding optimal index strategies (see [26] for a thorough introduction to the literature).

In 1985, the field spawned a new line of investigation with the paper [34] which introduced a new class of policies based on the principle of upper confidence bounding. While there are many versions of these policies, a standard one is

$$\nu_x^{UCB,n} = \theta_x^n + 4\sigma^W\sqrt{\frac{\log n}{N_x^n}}, \tag{46}$$

where θ_x^n is our estimate of the value of alternative x, and N_x^n is the number of times we evaluate alternative x within the first n iterations. The coefficient $4\sigma^W$ has a theoretical basis, but is typically replaced with a tunable parameter θ^{UCB} which we might write as

$$\nu_x^{UCB,n}(\theta^{UCB}) = \theta_x^n + \theta^{UCB}\sqrt{\frac{\log n}{N_x^n}}. \tag{47}$$

We would then use (43) to tune θ^{UCB} (this means we are searching for the best policy within this class). It is easy to see that the policy defined by (47) is much easier to compute than those based on Gittins indices. Just as the applied probability community pursued numerous variations of index policies, the machine learning community (in computer science)

has pursued a wide variety of UCB-based policies for variations of bandit problems (see [9] for an excellent review).

Another policy that has worked quite well in practice is interval estimation [30]

$$\nu_x^{IE,n}(\theta^{IE}) = \theta_x^n + \theta^{IE}\bar{\sigma}_x^n, \tag{48}$$

where $\bar{\sigma}_x^n$ is the standard deviation of the estimate θ_x^n. Again, θ^{IE} has to be tuned using (43), although we note that θ^{IE} is unitless, while this is not the case with θ^{UCB} (and a number of similar policies). Interestingly, it appears that interval estimation does not enjoy any of the theoretical bounds that have attracted so much attention from the computer science literature, yet it works extremely well in practice when properly tuned ([52] reports on a series of comparisons between interval estimation, UCB and knowledge gradient policies). See [61] for extensive experiments using MOLTE, a public domain testing environment for optimal learning.

The knowledge gradient policy, which maximizes the marginal value of information, was originally developed for offline (terminal reward) problems as we showed earlier. [52] developed a version for online (cumulative reward) problems, given by

$$\nu_x^{OLKG,n} = \bar{F}^n(x|\theta^n) + \tau\nu_x^{KG,n}, \tag{49}$$

where τ reflects a planning horizon. Here, the term $\tau\nu_x^{KG,n}$ captures the value of information gained from making, and learning from, a decision x, which plays the same role of encouraging exploration as the bonus terms in the Gittins index policy (45) or the UCB policy (46). The online KG policy would be written

$$X^{OLKG}(S^n) = \arg\max_x \left(\bar{F}^n(x|\theta^n) + \tau\nu_x^{KG,n}\right). \tag{50}$$

Note that all the policies for optimizing cumulative rewards include a current estimate of the reward we would receive at time n (either θ_x^n or $\bar{F}^n(x|\theta)$), along with another term that encourages exploration. Of these, the knowledge gradient policy is the only one with a clear delineation between a policy that uses pure learning for the offline case (since there are no rewards earned each time), and one that balances current rewards and value of information for the online case with cumulative rewards.

The popularity of index policies (based on Gittins index theory or the UCB index policies) has attracted considerable attention to search problems using the "bandit" vocabulary. Variations include names such as restless bandits [36] (learning problems where the underlying function is changing), intermittent bandits [18] (where alternatives are not always available), finite-horizon bandits [43], and contextual bandits [9] (where we are first given an initial state before observing our reward), to name just a few of the variants. This literature has been growing beyond its origins where alternatives (bandits) are discrete to include continuous-armed bandits [1] (now x can be continuous), and X-armed bandits [10] (where x can be a vector). The original problem used a lookup table belief model (where there was a belief θ_x^n about each discrete alternative) but this has been extended to linear belief models (so-called linear bandits [11], [57]) as well as to general response surfaces (response surface bandits [25]).

While writing this tutorial, we came to realize that there did not seem to be a formal definition of a "bandit" problem. Despite its introduction as a well defined problem (given in section 4.4), today a bandit problem is any sequential learning problem (which means the state S^n includes a belief state about the function $\mathbb{E}F(x,W)$) where we control the decisions of where to evaluate $F(x,W)$. At this stage we believe it is fair to say that the "bandit" vocabulary has outgrown its original motivating problem (slot machines).

We closed the previous section on the terminal reward case by noting that the challenge there was to find the best policy for collecting information to find a decision $x^{\pi,N}$. For the cumulative reward case, we no longer have the final step of using the information we have collected to make a deterministic decision. Instead, each of the decisions we make along the way matters, so the end result is the policy used to make all the decisions along the way.

6.5. Dynamic programming

We now turn our attention to problems where the state S_t includes a physical state. This is what is most commonly assumed when people use terms such as "dynamic programming" (or optimal control), although we recognize that pure learning problems can be formulated and solved as dynamic programs (we are unaware of learning problems being addressed within the optimal control community).

Consider a slight generalization of our newsvendor problem where left-over inventory is held until the next day. If R_t is the inventory at time t (that is, the inventory left over after satisfying demands between $t-1$ and t), the inventory equation would be

$$R_{t+1} = \max\{0, R_t + x_t - \hat{D}_{t+1}\}, \tag{51}$$

where x_t is the amount of new product ordered at time t (which arrives right away) and $\hat{D}_{t+1}$ is the demand that arises between t and $t+1$. Also assume that the price that we can sell our product evolves randomly according to

$$p_{t+1} = \theta_0 p_t + \theta_1 p_{t-1} + \varepsilon_{t+1}. \tag{52}$$

Thus, our transition function $S_{t+1} = S^M(S_t, x_t, W_{t+1})$ is given by equations (51) - (52), where $W_{t+1} = (\hat{D}_{t+1}, \varepsilon_{t+1})$. Our state variable is $S_t = (R_t, p_t, p_{t-1})$. The contribution function is given by

$$C(S_t, x_t, \hat{D}_{t+1}) = p_t \min\{R_t + x_t, \hat{D}_{t+1}\} - cx_t.$$

Let $X_t^\pi(S_t)$ be the policy that determines the order quantity x_t given the state variable S_t, where the policy has to return decisions that fall within the constraint set $\mathcal{X}_t$ (which may depend on S_t). Our objective function would be written

$$\max_\pi \mathbb{E}\left\{\sum_{t=0}^{T} C(S_t, X_t^\pi(S_t), W_{t+1}) | S_0\right\}, \tag{53}$$

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

This is a classical dynamic program where the state variable consists of an endogenously controlled physical state R_t, and an exogenously controlled price process p_t. It is useful to compare our dynamic program (53) with the offline (terminal reward) and online (cumulative reward) versions of our sequential stochastic search problems, which we repeat below for convenience:

$$\text{Online:} \quad \max_\pi \mathbb{E}\left\{\sum_{n=0}^{N-1} F(X^\pi(S^n), W^{n+1}) | S_0\right\}, \tag{54}$$

$$\text{Offline:} \quad \max_\pi \mathbb{E}\{\max_x F(x^{\pi,N}, W) | S_0\}. \tag{55}$$

We see the obvious similarity between our dynamic program (53) and the (online) stochastic search with cumulative rewards (54). The similarity is notable because dynamic programs are not generally thought of as instances of online learning problems. In practice, the vast majority of dynamic programs are solved "offline" where we use some algorithm to find a policy that can then be used "online." At the same time, readers will also see that the terminal cost objective (55) and the cumulative cost objective (54) are mathematically equivalent to our statement of a "dynamic program" in (53). We can convert the terminal-cost objective to (53) by simply setting

$$C_t(S_t, X^\pi(S_t), W_{t+1}) = \begin{cases} 0 & t < T \\ F(x^{\pi,T}, W) & t = T. \end{cases}$$

For the cumulative cost criterion, we would write

$$C_t(S_t, X^\pi(S_t), W_{t+1}) = F(X^\pi(S_T), W_{n+1}),\ t = 0, \ldots, T,$$

where we have made the conversion from iteration n to time t.

This is not an idle observation. There is an entire class of algorithmic strategies that have evolved under names such as reinforcement learning and approximate (or adaptive) dynamic programming which make decisions by simulating repeatedly through the planning horizon. These strategies could be implemented in either offline or online settings. If we are in state S_t^n at iteration n at time t, we can get a sampled estimate of the value of being in the state using

$$\hat{v}_t^n = \big(C(S_t^n, x_t^n) + \mathbb{E}\{\overline{V}_{t+1}(S_{t+1})|S_t^n, x_t^n\}\big).$$

We can then use this decision to update our estimate of the value of being in state S_t^n using

$$\overline{V}_t^n(S_t^n) = (1-\alpha_n)\overline{V}_t^{n-1}(S_t^n) + \alpha_n \hat{v}_t^n.$$

To use this updating strategy we need a policy for selecting states. Typically, we specify some policy $X^\pi(S_t^n)$ to generate an action x_t^n from which we sample a downstream state using $S_{t+1}^n = S^M(S_t^n, x_t^n, W_{t+1}(\omega^n))$. A simple greedy policy would be to use

$$X^\pi(S_t^n) = \arg\max_{x_t \in \mathcal{X}_t} \big(C(S_t^n, x_t) + \mathbb{E}\{\overline{V}_{t+1}(S_{t+1})|S_t^n, x_t\}\big). \tag{56}$$

This is known as a *pure exploitation policy* in the optimal learning literature [47], which only produces good results in special cases such as when we can exploit convexity in the value function ([53],[40]). Consider the policies developed for discrete alternatives for online (cumulative cost) learning problems that we introduced in section 6.4 such as the Gittins index policy (45) or upper confidence bounding (46). Both of these policies have the structure of choosing an action based on an immediate cost (which in this problem consists of both the one step contribution $C(S_t, x_t)$ and the downstream value $\mathbb{E}\{V_{t+1}(S_{t+1})|S_t\}$), to which is added some form of "uncertainty bonus" which encourages exploration.

There is a wide range of heuristic policies that have been suggested for dynamic programming that ensure sufficient exploration. Perhaps the most familiar is epsilon-greedy, which chooses a greedy action (as in (56)) with probability ϵ, and chooses an action at random with probability $1-\epsilon$. Others include Boltzmann-exploration (where actions are chosen with probabilities based on a Boltzmann distribution), and strategies with names such as E^3 [32] and R-max [7]. However, the depth of research for learning when there is a physical state does not come close to the level of attention that has been received for pure learning problems.

[51] develops knowledge gradient policies for both online learning (maximizing cumulative rewards) as well as offline learning (maximizing final reward). The online policy (cumulative reward) has the structure

$$x_t^n = \arg\max_{x_t \in \mathcal{X}_t} \big(C(S_t^n, x_t) + \mathbb{E}\{\overline{V}_{t+1}(S_{t+1})|S_t^n, x_t\} + \mathbb{E}\{\nu_x^{KG,n}(S_t^{x,n}, S_{t+1})|S_t^x\}\big). \tag{57}$$

Here, $\nu_x^{KG,n}(S_t^{x,n}, S_{t+1})$ is the value of information derived from being in post-decision state $S_t^{x,n}$ (the state produced by being in state S_t^n and taking action x_t) and observing the information in the random transition from $S_t^{x,n}$ to the next pre-decision state S_{t+1}.

The corresponding policy for offline learning (maximizing the final reward at time N) proposed in [51] is given by

$$x_t^n = \arg\max_{x_t \in \mathcal{X}_t} \mathbb{E}\{\nu_x^{KG,n}(S_t^{x,n}, S_{t+1})|S_t^x\}. \tag{58}$$

	Offline Terminal reward	Online Cumulative reward
Learning problems	$\max_\pi \mathbb{E}F(X^{\pi,N},W)$ Stochastic search	$\max_\pi \mathbb{E}\sum_{n=0}^{N-1} F(X^\pi(S^n),W^{n+1})$ Multiarmed bandit problem
Control problems	$\max_{\pi^{learn}} \mathbb{E}\sum_{t=0}^{T} C(S_t,X^{\pi^{impl}}(S_t))$ Dynamic programming	$\max_\pi \mathbb{E}\sum_{t=0}^{T} C(S_t,X^\pi(S_t))$ Dynamic programming

TABLE 1. Comparison of formulations for learning vs. control problems, and offline (terminal reward) and online (cumulative reward).

Not surprisingly, the policies for online (cumulative reward) and offline (final reward) learning for dynamic programming (with a physical state) closely parallel with our online (54) and offline (55) policies for pure learning.

So this raises a question. We see that the policy (54) is designed for the (online) cumulative reward objective (43), while the policy (55) is designed for the (offline) final-reward objective. We have further argued that our generic objective function for a dynamic program (53) closely parallels the (online) cumulative reward objective (43). The policy (57) balances exploitation (the contribution plus value term) and exploration (the value of information term), which is well suited to learning for (online) cumulative reward problems. Given all this, we should ask, what is the objective function that corresponds to the knowledge gradient policy for (offline) final-reward learning for dynamic programs?

The answer to this question lies in looking carefully at the (offline) final-reward objective for stochastic search given in (39). This problem consists of looking for a policy that learns the best value for the decision x after the learning budget is exhausted. We can designate the "policy" as a *learning policy*, while x is the *implementation decision*. For our fully sequential dynamic program, the policy (58) is a learning policy π^{learn}, but to learn what? The answer is that we are learning an *implementation policy* π^{impl}. That is, we are going to spend a budget using our learning policy (58), where we might be learning value functions or a parametric policy function (see section 8 for further discussions of how to construct policies).

Stated formally, the sequential version of the offline final-reward objective (39) can be written

$$\max_{\pi^{learn}} \mathbb{E}\left\{\sum_{t=0}^{T} C(S_t,X^{\pi^{impl}}(S_t))|S_0\right\}. \tag{59}$$

We note that the implementation policy $X^{\pi^{impl}}$ is a function of the learning policy. Table 1 shows all four problems divided by learning vs. control problems (by which we mean sequential problems with a physical state), and offline (terminal reward) and online (cumulative reward) objectives.

7. Some extensions

We are going to take a brief detour through two important problem classes that are distinctly different, yet closely related. The first involves learning in the presence of dynamic, exogenous state information that produces a problem known under various names, but one is *contextual bandits*, where each time we need to make a decision, we are handed a different state of the world. The second involves building bridges to the entire field of statistical learning.

7.1. Stochastic search with exogenous state information

There are many problems where information is first revealed, after which we make a decision, and then more information is revealed. Using our newsvendor example, we might first see the weather (or a weather forecast), then we have to make a decision, and then we finally

see the demand (which might depend on the weather). Another example arises in a health setting where a patient arrives for treatment, and a doctor has to make treatment decisions. The attributes of the patient represent initial information that is revealed exogenously, then a decision is made, followed by a random outcome (the success of the treatment).

In both of these examples, we have to make our decision given advance information (the weather, or the attributes of the patient). We could write this as a standard stochastic search problem, but conditioned on a dynamic initial state S_0 which is revealed each time before solving the problem, as in

$$\max_x \mathbb{E}\{F(x,W)|S_0\}. \tag{60}$$

Instead of finding a single optimal solution x^*, we need to find a function $x^*(S_0)$. This function is a form of policy (since it is a mapping of state to action). This is known in the bandit literature as a *contextual bandit* problem [9]; however this literature has not properly modeled the full dynamics of this problem.

We propose the following model. First, we let K_t be our "state of knowledge" at time t that captures our belief about the function $F(x) = \mathbb{E}F(x,W)$ (keep in mind that this is distributional information). We then model two types of exogenous information. The first we call W_t^e which is exogenous information that arrives before we make a decision (this would be the weather in our newsvendor problem, or the attributes of the patient before making the medical decision). Then, we let W_{t+1}^o be the exogenous information that captures the outcome of the decision after the decision x_t. The exogenous outcome W_t^o, along with the decision x_t and the information (K_t and W_t^e), is used to produce an updated state of knowledge K_{t+1}.

Using this notation, the sequencing of information, knowledge states and decisions is

$$K_0, W_0^e, x_0, W_1^o, K_1, W_1^e, x_1, W_2^o, K_2, \ldots.$$

We have written the sequence (W_t^o, K_t, W_t^e) to reflect the logical progression where we first learn the outcome of a decision W_t^o, then update our knowledge state producing K_t, and then observe the new exogenous information W_t^e before making decision x_t. However, we can write $W_t = (W_t^o, W_t^e)$ as the exogenous information, which leads to a new state $S_t = (K_t, W_t^e)$. Our policy $X_t^\pi(S_t)$ will depend on both our state of knowledge K_t about $\mathbb{E}F(x,W)$, as well as the new exogenous information. This change of variables, along with defining $S_0 = (K_0, W_0^e)$, gives us our standard sequence of states, actions and new information, with our standard search over policies (as in (54)) for problems with cumulative rewards.

There is an important difference between this problem and the original terminal reward problem. In that problem, we had to find the best policy to collect information to help us make a deterministic decision, $x^{\pi,N}$. When we introduce the exogenous state information, it means we have to find a policy to collect information, but now we are using this information to learn a function $x^{\pi,N}(W^e)$ which we recognize is a form of policy. This distinction is less obvious for the cumulative reward case, where instead of learning a policy $X^\pi(K^n)$ (when $S^n = K^n$), we are now learning a function (policy) $X^\pi(K^n, W^e)$ with an additional variable.

Thus, we see again that a seemingly new problem class is simply another instance of a sequential learning problem.

7.2. From stochastic optimization to statistical learning

There are surprising parallels between stochastic optimization and statistical learning, suggesting new paths for unification. Table 2 compares a few problems, starting with the most basic problem in row (1) in statistics of fitting a specified model to a batch dataset. Now contrast this to an instance of the sample average approximation on the right. In row (2) we have an instance of an online learning problem where data (in the form of pairs (Y, X))

	Statistical learning	Stochastic optimization
(1)	Batch estimation: $\min_\theta \frac{1}{N}\sum_{n=1}^{N}(y_n - f(x_n\|\theta))^2$	Sample average approximation: $x^* = \arg\max_{x\in\mathcal{X}} \frac{1}{N}F(x, W(\omega^n))$
(2)	Online learning: $\min_\theta \mathbb{E}F(Y - f(X\|\theta))^2$	Stochastic search: $\min_\theta \mathbb{E}F(X, W)$
(3)	Searching over functions: $\min_{f\in\mathcal{F},\theta\in\Theta^f} \mathbb{E}F(Y - f(X\|\theta))^2$	Policy search: $\min_\pi \mathbb{E}\sum_{t=0}^{T} C(S_t, X^\pi(S_t))$

TABLE 2. Comparison of classical problems faced in statistics (left) versus similar problems in stochastic optimization (right).

arrive sequentially, and we have to execute one step of an algorithm to update θ. On the right, we have a classical stochastic search algorithm which we can execute in an online fashion using a stochastic gradient algorithm.

Finally, row (3) restates the estimation problem but now includes the search over functions, as well as the parameters associated with each function. On the right, we have our (now familiar) optimization problem where we are searching over policies (which is literally a search over functions). We anticipate that most researchers in statistical machine learning are not actually searching over classes functions, any more than the stochastic optimization community is searching over classes of policies. However, both communities aspire to do this.

We see that the difference between statistical learning and stochastic optimization is primarily in the nature of the cost function being minimized. Perhaps these two fields should be talking more?

8. Designing policies

We have shown that a wide range of stochastic optimization problems can be formulated as sequential decision problems, where the challenge is to solve an optimization problem over policies. We write the canonical problem as

$$\max_\pi \mathbb{E}\left\{\sum_{t=0}^{T} C(S_t, X_t^\pi(S_t))|S_0\right\}, \tag{61}$$

where $S_{t+1} = S^M(S_t, X_t^\pi(S_t), W_{t+1})$. We now have to address the challenge: how do we search over policies?

There are two fundamental strategies for designing effective (and occasionally optimal) policies:

Policy search - Here we search over a typically parameterized class of policies to find the policy (within a class) that optimizes (61).
Policies based on lookahead approximations - These are policies that are based on approximating the impact of a decision now on the future.

Both of these strategies can lead to optimal policies in very special cases, but these are rare (in practice) and for this reason we will assume that we are working with approximations.

8.1. Policy search

Policy search involves searching over a space of functions to optimize (61). This is most commonly done when the policy is a parameterized function. There are two approaches to representing these parameterized functions:

Optimizing a policy function approximation (PFA) - The analytical function might be a linear function (where it is most commonly known as an *affine policy*), a nonlinear function, or even a locally linear function. An affine policy might be as simple as

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

where $(\phi_1(S_t), \phi_2(S_t))$ are features extracted from the state variable S_t. Policy search means using (61) to search for the best value of θ.

Optimizing a parametric cost function approximation (CFA) - Consider a policy of the form

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

This represents a myopic policy, although we could use a deterministic lookahead as an approximation. We may replace the contribution $C(S_t, x_t)$ with an appropriately modified contribution $\bar{C}^\pi(S_t, x_t|\theta)$; in addition, we might replace the constraints $\mathcal{X}_t$ with modified constraints $\mathcal{X}_t^\pi(\theta)$. The policy would be written

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

For example, we might use a modified cost function with an additive correction term

$$X^\pi(S_t|\theta) = \arg\max_{x_t \in \mathcal{X}_t^\pi(\theta)} \Big(C(S_t, x_t) + \sum_{f \in \mathcal{F}} \theta_f \phi_f(S_t, x_t)\Big). \tag{62}$$

We might replace constraints of the form $Ax = b,\ x \le u$ with $Ax = b,\ x \le u + D\theta$ where the adjustment $D\theta$ has the effect of introducing buffer stocks or schedule slack. This is how uncertainty is often handled in industrial applications, although the adjustment of θ tends to be very ad hoc.

Once we have our parameterized function $X_t^\pi(S_t|\theta)$, we use classical stochastic search techniques to optimize θ by solving

$$\max_\theta \mathbb{E}\left\{\sum_{t=0}^{T} C(S_t, X_t^\pi(S_t|\theta))|S_0\right\}. \tag{63}$$

We note that this is the same as the stochastic search problem (9) which, as we have pointed out earlier, can also (typically) be formulated as a dynamic program (or more precisely, as an offline sequential learning problem).

Policy function approximations can be virtually any statistical model, although lookup tables are clumsy, as are nonparametric models (although to a lesser degree). Locally parametric models have been used successfully in robotics [20], although historically this has tended to require considerable domain knowledge.

Policy search typically assumes a particular structure: a linear or nonlinear model, perhaps a neural network. Given a structure, the problem reduces to searching over a well-defined parameter space that describes the class of policies. Search procedures are then divided between derivative-based, and derivative-free. Derivative-based methods assume that we can take the derivative of

$$F(\theta, \omega) = \sum_{t=0}^{T} C(S_t(\omega), X_t^\pi(S_t(\omega))), \tag{64}$$

where $S_{t+1}(\omega) = S^M(S_t(\omega), X_t^\pi(S_t(\omega)), W_{t+1}(\omega))$ represents the state transition function for a particular sample path ω. If these sample gradients are available, we can typically tackle high-dimensional problems. However, there are many problems which require derivative-free

search, which tends to limit the complexity of the types of policies that can be considered. For example, a particularly problematic problem class is time-dependent policies.

Policy search can produce optimal policies for special classes where we can identify the structure of an optimal policy. One of the most famous is known as (q, Q) inventory policies, where orders are placed with the inventory $R_t < q$, and the order $x_t = Q - R_t$ brings the inventory up to Q. However, the optimality of this famous policy is only for the highly stylized problems considered in the research literature.

8.2. Policies based on lookahead approximations

While policy search is an exceptionally powerful strategy for problems where the policy has structure that can be exploited, there are many problems where it is necessary to fall back on the more brute-force approach of optimizing over the entire horizon starting with the current state.

We develop this idea by starting with the realization that we can characterize the optimal policy for any problem using

$$X_t^*(S_t) = \arg\min_{x_t} \left(C(S_t, x_t) + \min_{\pi \in \Pi} \mathbb{E} \left\{ \sum_{t'=t+1}^{T} C(S_{t'}, X_{t'}^{\pi}(S_{t'})) \middle| S_t, x_t \right\} \right). \tag{65}$$

The problem with equation (65) is that it is impossible to solve for the vast majority of problems (although we note that this is the same as solving a decision tree). The difficulties in solving the problem starting at time t are the same as when we start at time 0: we cannot compute the expectation exactly, and we generally do not know how to optimize over the space of policies.

For this reason, the research community has developed two broad strategies for approximating lookahead models:

Value function approximations (VFA) - Widely known as approximate dynamic programming or reinforcement learning, value function approximations replace the lookahead model with a statistical model of the future that depends on the downstream state resulting from starting in state S_t and making decision x_t.

Approximations of the lookahead model - Here, we approximate the model itself to make equation (65) computationally tractable.

We describe these in more detail in the following subsections.

8.2.1. Value function approximations Bellman first introduced the idea of capturing the value of being in a state using his famous optimality equation

$$\begin{aligned} V_t(S_t) &= \min_{a_t} \left(C(S_t, a_t) + \min_{\pi \in \Pi} \mathbb{E} \left\{ \sum_{t'=t+1}^{T} C(S_{t'}, X_{t'}^{\pi}(S_{t'})) \middle| S_t \right\} \right), \\ &= \min_{a_t} \left[C(S_t, a_t) + \mathbb{E} \left\{ \min_{a_{t+1}} \left[C(S_{t+1}, a_{t+1}) + \mathbb{E} \left\{ \min_{a_{t+2}} \dots | S_{t+1}, a_{t+1} \right\} \right] | S_t, a_t \right\} \right], \\ &= \min_{a_t} \left[C(S_t, a_t) + \mathbb{E}\{ V_{t+1}(S_{t+1}) | S_t, a_t \} \right]. \end{aligned} \tag{66}$$

We write this equation assuming that $S_{t+1} = S^M(S_t, a_t, W_{t+1})$ (note the similarities with our multistage stochastic program (23)). The Markov decision process community prefers to compute

$$P(S_{t+1} = s' | S_t = s, a_t) = \mathbb{E}\{ \mathbb{1}_{\{S^M(S_t, a_t, W_{t+1}) = s'\}} \}.$$

Using this matrix produces the more familiar form of Bellman's equation given in equation (24). This community often treats the one-step transition matrix as data, without realizing

that this is often a computationally intractable function. It is well-known that state variables are often vectors, producing the well-known curse of dimensionality when using what is known as a flat representation, where states are numbered $1,\ldots,|\mathcal{S}|$. It is often overlooked that the random information W_t may be a vector (complicating the expectation), in addition to the action a_t (which the operations research community writes as a vector x_t). These make up the three curses of dimensionality.

Bellman's equation works well for problems with small state and action spaces, and where the transition matrix can be easily computed. There are real problems where this is the case, but they are small. For example, there are many problems with small state and action spaces, but where the random variable W_t can only be observed (the distribution is unknown). There are many other problems where the state variable has more than three or four dimensions, or where they are continuous. Of course, there are many problems with vector-valued decisions x_t.

To deal with the potentially three curses of dimensionality, communities have evolved under names such as approximate dynamic programming ([55], [45]), reinforcement learning ([59], [60]), and adaptive dynamic programming. While these terms cover a host of algorithms, there are two broad strategies that have evolved for estimating value functions. The first, known as approximate value iteration, involves bootstrapping a current value function approximation where we would calculate

$$\hat{v}_t^n = \max_{a_t}\big(C(S_t^n,a_t)+\mathbb{E}\{\overline{V}_{t+1}^{n-1}(S_{t+1}^n)|S_t^n\}\big). \tag{67}$$

We then use $\hat{v}_t^n$ to update our value function approximation. For example, if we are using a lookup table representation, we would use

$$\overline{V}_t^n(S_t^n) = (1-\alpha)\overline{V}_t^{n-1}(S_t^n)+\alpha\hat{v}_t^n.$$

We simplify the process if we use the post-decision state S_t^a, which is the state after an action a_t is taken, but before new information has arrived. We would calculate $\hat{v}_t^n$ using

$$\hat{v}_t^n = \max_{a_t}\big(C(S_t^n,a_t)+\overline{V}_{t+1}^{a,n-1}(S_t^{a,n})\big). \tag{68}$$

$\hat{v}_t^n$ is a sample estimate of the value of being at pre-decision state S_t^n. Thus, we have to step back to the previous post-decision state $S_{t-1}^{a,n}$, which we do using

$$\overline{V}_{t-1}^n(S_{t-1}^{a,n}) = (1-\alpha)\overline{V}_{t-1}^{n-1}(S_{t-1}^{a,n})+\alpha\hat{v}_t^n.$$

An alternative approach for calculating $\hat{v}_t^n$ involves simulating a suboptimal policy. For example, we may create a VFA-based policy using

$$X_t^{VFA,n}(S_t) = \arg\min_{a_t}\big(C(S_t,a_t)+\overline{V}_t^{a,n-1}(S_t^a)\big), \tag{69}$$

Now consider simulating this policy over the remaining horizon using a single sample path ω^n, starting from a state S_t^n. This gives us

$$\hat{v}_t^n = \sum_{t'=t}^{T} C(S_{t'}^n(\omega^n), A_{t'}^{VFA,n}(S_{t'}^n(\omega^n))).$$

Often this is calculated as a backward pass using

$$\hat{v}_t^n = C(S_t^n(\omega^n), A_t^{VFA,n}(S_t^n(\omega^n)))+\hat{v}_{t+1}^n, \tag{70}$$

where $\hat{v}^n_{t+1}$ is calculated starting at $S^n_{t+1} = S^M(S^n_t, A^{VFA,n}_t(S^n_t(\omega^n)))$. We note only that either the forward pass approach (68) or the backward pass approach (70) may be best for a particular situation.

The approximation strategy that has attracted the most attention in the ADP/RL literature, aside from lookup tables, has been the use of linear architectures of the form

$$\overline{V}_t(S_t) = \sum_{f\in\mathcal{F}} \theta_f \phi_f(S_t), \tag{71}$$

where $(\phi_f(S_t))_{f\in\mathcal{F}}$ is a set of features that have to be chosen. We note that if we substitute the linear value function approximation (71) into the policy (69), we get a policy that looks identical to our parametric cost function approximation in (62). In fact, it is not unusual for researchers to begin with a true value function approximation (71) which is estimated using updates that are calculated using forward (68) or backward (70) passes. Once they get an initial estimate of the parameter vector θ, they can use policy search to further tune it (see [37] for an example). However, if we use policy search to tune the coefficient vector, then the linear model can no longer be viewed as a value function approximation; now, it is a cost function approximation.

These techniques are also used for multistage linear programs. In this setting, we would use gradients or dual variables to build up convex approximations of the value functions. Popular methods are based on Benders cuts using a methodology known as the stochastic dual decomposition procedure (SDDP) ([44], [53], [54]). An overview of different approximation methods are given in [45][Chapter 8].

8.2.2. Direct lookahead models All the strategies described up to now (PFAs, CFAs, and VFAs) have depended on some form of functional approximation. This tends to work very well when we can exploit problem structure to develop these approximations. However, there are many applications where this is not possible. When this is the case, we have to resort to a policy based on a direct lookahead.

Assuming that we cannot solve the base model (65) exactly (which is typically the case), we need to create an approximation that we call the *lookahead model*, where we have to introduce approximations to make them more tractable. We are able to identify five types of approximations that can be used when creating a lookahead model (this is taken from [46]):

Limiting the horizon - We may reduce the horizon from (t,T) to $(t,t+H)$, where H is a suitable short horizon that is chosen to capture important behaviors. For example, we might want to model water reservoir management over a 10 year period, but a lookahead policy that extends one year might be enough to produce high quality decisions. We can then simulate our policy to produce forecasts of flows over all 10 years.

Stage aggregation - A stage represents the process of revealing information followed by the need to make a decision. A common approximation is a two-stage formulation, where we make a decision x_t, then observe all future events (until $t+H$), and then make all remaining decisions. A more accurate formulation is a multistage model, but these can be computationally very expensive.

Outcome aggregation or sampling - Instead of using the full set of outcomes Ω (which is often infinite), we can use Monte Carlo sampling to choose a small set of possible outcomes that start at time t (assuming we are in state S^n_t during the nth simulation through the horizon) through the end of our horizon $t+H$. The simplest model in this class is a deterministic lookahead, which uses a single point estimate.

Discretization - Time, states, and decisions may all be discretized in a way that makes the resulting model computationally tractable. In some cases, this may result in a Markov decision process that may be solved exactly using backward dynamic programming (see [48]). Because the discretization generally depends on the current state S_t, this model will have to be solved all over again after we make the transition from t to $t+1$.

Dimensionality reduction - We may ignore some variables in our lookahead model as a form of simplification. For example, a forecast of weather or future prices can add a number of dimensions to the state variable. While we have to track these in the base model (including the evolution of these forecasts), we can hold them fixed in the lookahead model, and then ignore them in the state variable (these become *latent variables*).

We illustrate a notational system for lookahead models. First, all variables are indexed by time t' where t is the time at which we are creating and solving the lookahead model, and t' indexes time within the lookahead model.

To avoid confusion with our base model, we use the same variables as the base model but use tilde's to identify when we are modeling the lookahead model. Thus, $\tilde{S}_{tt'}$ is the state at time t' in the lookahead model (created at time t), $\tilde{x}_{tt'}$ is the decision variable, and $\tilde{W}_{tt'}$ is our exogenous information that first becomes known at time t' within the lookahead model (if we are using a stochastic lookahead). The sample paths $\tilde{\omega}_t \in \tilde{\Omega}_t$ are created on the fly. We write our transition function as

$$\tilde{S}_{t,t'+1} = \tilde{S}^M(\tilde{S}_{tt'}, \tilde{x}_{tt'}, \tilde{W}_{t,t'+1}(\tilde{\omega}_t)),$$

where $\tilde{S}^M(\cdot)$ is the appropriately modified version of the transition function for the base model $S^M(\cdot)$.

The simplest lookahead model would use a point forecast of the future, where we might write $\overline{W}_{tt'} = \mathbb{E}\{W_{t'}|S_t\}$. We would write a deterministic lookahead policy as

$$X_t^{LA-D}(S_t|\theta) = \arg\max_{\tilde{x}_{tt}} \left(\max_{\tilde{x}_{t,t+1},\ldots,\tilde{x}_{t,t+H}} \sum_{t'=t}^{t+H} C(\tilde{S}_{tt'}, \tilde{x}_{tt'}) \right). \tag{72}$$

Here, we use θ to capture all the parameters that characterize our lookahead model (horizon, discretization, sampling, staging of information and decisions).

A stochastic lookahead model can be created using our sampled set of outcomes $\tilde{\Omega}_t^n$, giving us a stochastic lookahead policy

$$X_t^{LA-SP,n}(S_t^n) = \arg\max_{x_t} \left(C(S_t^n, x_t) + \min_{(\tilde{x}_{tt'}(\tilde{\omega}),\ldots,\tilde{x}_{t,t+H}(\tilde{\omega})),\forall \tilde{\omega} \in \tilde{\Omega}_t^n} \tilde{\mathbb{E}}^n \left\{ \sum_{t'=t+1}^{t+H} C(\tilde{S}_{tt'}, \tilde{x}_{tt'}(\tilde{\omega})) \middle| S_t, x_t \right\} \right). \tag{73}$$

This model is basically a sampled approximation of the multistage stochastic program given in (23).

If the actions are discrete, but where the exogenous information is complex, we can use a technique called Monte Carlo tree search to search the tree without enumerating it. [8] provides a survey, primarily in the context of MCTS for deterministic problems (which have received the most attention in the computer science community where this idea has been developed). [16] introduces a method known as "double progressive widening" to describe an adaptation of classical MCTS for stochastic problems.

When decisions are vectors, we can turn to the field of stochastic programming. Either the two-stage stochastic program introduced in section 4.6 or the multistage stochastic program from section 4.7 can be used to create approximate lookahead models. At the heart of this methodology is separating a controllable resource state R_t from an exogenous information process I_t. Stochastic linear programming exploits the fact that the problem is convex in R_t to build powerful and highly effective convex approximations, but has to resort to sampled versions of the information process in the form of scenario trees. See ([6], [31], [54]) for excellent introductions to the field of stochastic linear programming.

	Deterministic model	Stochastic model
Objective function	$\min_{x_0,\ldots,x_T} \sum_{t=0}^{T} c_t x_t$	$\max_\pi \mathbb{E}\left\{\sum_{t=0}^{T} C(S_t, X_t^\pi(S_t))\right\}$
Decisions	$x_0, \ldots, x_T$	Policy $X^\pi : \mathcal{S} \to \mathcal{X}$
Constraints at t	$\mathcal{X}_t = \{x \vert A_t x_t = R_t, x_t \geq 0\ \}$	$X^\pi(S_t) \in \mathcal{X}_t$
Transition function	$R_{t+1} = b_{t+1} + B_t x_t$	$S_{t+1} = S^M(S_t, x_t, W_{t+1})$
Exogenous inf.		$W_t, \ldots, W_T \in \Omega$

TABLE 3. Comparison of the elements of a (time-staged) deterministic linear program (left) and a sequential decision process (dynamic program) (right).

8.3. Remarks

We note that these two strategies (policy search and lookahead approximations) which combine to create four classes of policies (policy function approximations, parametric cost function approximations, value function approximations and lookahead models), span *all* the strategies that we have ever seen in any of the communities of stochastic optimization listed in the beginning of this article.

It is important to keep in mind that while the goal in a deterministic optimization problem is to find a decision x (or a, or u), the goal in a stochastic problem is to find a policy $X^\pi(S)$ (or $A^\pi(S)$ or $U^\pi(S)$), which has to be evaluated in the objective function given by (61). Our experience has been that people who use stochastic lookahead models (which can be quite hard to solve) often overlook that these are just policies for solving a stochastic base model such as that shown in (61).

Figure 3 shows a side by side comparison of a generic time-staged deterministic linear program and a corresponding formulation of a sequential decision problem (that is, a dynamic program).

9. Uncertainty modeling

Just as important as designing a good policy is using a good model of the uncertainties that affect our system. If we do not accurately represent the nature of uncertainty that we have to handle, then we are not going to be able to identify good policies for dealing with uncertainty.

Below we provide a brief overview of the types of uncertainty that we may want to incorporate (or at least be aware of), followed by a discussion of state-dependent information processes.

9.1. Types of uncertainty

Uncertainty is communicated to our modeling framework in two ways: the initial state S_0 which is used to capture probabilistic information about uncertain information (Bayesian priors), and the exogenous information process W_t. While beyond the scope of this chapter, below is a list of different mechanisms that describe how uncertainty can enter a model.

- Observational errors - This arises from uncertainty in observing or measuring the state of the system. Observational errors arise when we have unknown state variables that cannot be observed directly (and accurately).
- Prognostic uncertainty - Uncertainty in a forecast of a future event.
- Experimental noise - This describes the uncertainty that is seen in repeated experiments (this is distinctly different from observational uncertainty).
- Transitional uncertainty - This arises when we have a deterministic model of how our system should evolve as a result of an action or control, but errors are introduced that disturb the process.

- Inferential (or diagnostic) uncertainty - This is the uncertain in statistical estimates based on observational data.
- Model uncertainty - This may include uncertainty in the transition function, and uncertainty in the distributions driving the stochastic processes (e.g. uncertainty in the parameters of these distributions).
- Systematic uncertainty - This covers what might be called "state of the world" uncertainty, such as the average rate of global warming or long term price trends.
- Control uncertainty - This is where we choose a control u_t (such as a price, concentration of a chemical, or dosage of a medicine), but what happens is $\hat{u}_t = u_t + \delta u_t$ where δu_t is a random perturbation.
- Adversarial behavior - The exogenous information may be coming from an adversary, whose behavior is uncertain.

It is unlikely that a model will capture all of these different types of uncertainty, but we feel it is useful to at least be aware of them.

9.2. State dependent information processes

It is relatively common to view the exogenous information process as if it is purely exogenous. That is, let $\omega \in \Omega$ represent a sample path for $W_1, W_2, \ldots, W_T$. We often model these as if they are, or at least could be, generated in advance and stored in some file. Assume we do this, and let $p(\omega)$ be the probability that $\omega \in \Omega$ would happen (this might be as simple as $p(\omega) = 1/|\Omega|$). In this case, we would write our objective function as

$$\max_{\pi} \sum_{\omega \in \Omega} p(\omega) \sum_{t=0}^{T} C(S_t(\omega), X_t^{\pi}(S_t(\omega))),$$

where $S_{t+1}(\omega) = S^M(S_t(\omega), X_t^{\pi}(S_t(\omega)), W_{t+1}(\omega))$.

There are many problems where the exogenous information process depends on the state and/or the action/decision. We can write this generally by assuming that the distribution of W_{t+1} depends (at time t) on the state S_t or decision x_t, or more compactly, the post-decision state S_t^x. In this case, we would write our objective function as

$$\max_{\pi} \mathbb{E}^{\pi} \left\{ \sum_{t=0}^{T} C(S_t, X_t^{\pi}(S_t)) | S_0 \right\}.$$

The difference is that we have written $\mathbb{E}^{\pi}$ rather than $\mathbb{E}$ to reflect the fact that the sequence $W_1, W_2, \ldots, W_T$ depends on the post-decision state, which depends on what policy we are following. In this case, we would never generate the exogenous information sequences in advance; in fact, we suspect that this is generally not done even when W_t does not depend on the state of the system.

It is relatively straightforward to capture state-dependent information processes while simulating any of our policies. Let $P^W(W_{t+1} = w | S_t, x_t)$ be the conditional distribution of W_{t+1} given the state S_t and action x_t. As we simulate a policy $X_t^{\pi}(S_t)$, we use S_t to generate $x_t = X_t^{\pi}(S_t)$, and then sample W_{t+1} from the distribution $P^W(W_{t+1} = w | S_t, x_t)$. This does not cause any problems with policy search, or simulating policies for the purpose of creating value function approximations.

The situation becomes a bit more complex with stochastic lookahead models. The problem is that we generate a lookahead model before we have made the decision. It is precisely for this reason that communities such as stochastic programming assume that the exogenous information process has to be independent of decisions. While this is often viewed as a limitation of stochastic programming, in fact it is just one of several approximations that may be necessary when creating an approximate lookahead model. Thus, we re-emphasize

that the stochastic process in the lookahead model is designated $\tilde{W}_{tt'}$ for $t' = t, \ldots, t+H$ since this is distinct from the true information process $W_t, \ldots, W_T$ in the base model. The exogenous information process in the stochastic lookahead model will invariably involve simplifications; ignoring the dependence on the state of the system is one of them (but hardly the only one).

10. Closing remarks

We opened this article making the point that while deterministic optimization enjoys widely used canonical forms (linear/nonlinear/integer programming, deterministic control), stochastic optimization has been characterized with a diverse set of modeling frameworks with different notational systems, mathematical styles and algorithmic strategies. These are typically motivated by the wide diversity of problem settings that arise when we introduce uncertainty.

We have made the case that virtually all stochastic optimization problems, including stochastic search and even statistical learning, can be posed using a single canonical form which we write as

$$\max_{\pi} \mathbb{E}\left\{\sum_{t=0}^{T} C(S_t, X_t^{\pi}(S_t))|S_0\right\},$$

where decisions are made according to a policy $x_t = X_t^{\pi}(S_t)$, where we might use $a_t = A_t^{\pi}(S_t)$ for discrete actions, or $u_t = U_t^{\pi}(x_t)$ for state x_t and control u_t. States evolve according to a (possibly unknown) transition function $S_{t+1} = S^M(S_t, X_t^{\pi}(S_t), W_{t+1})$.

All of the modeling devices we use are drawn from the literature, but they are not widely known. The concept of a policy is not used at all in stochastic programming, and when it is used (in dynamic programming and control), it tends to be associated with very specific forms (policies based on value functions, or parameterized control laws). This is accompanied with widespread confusion about the nature of a state variable, for which formal definitions are rare.

While we believe it will always be necessary to adapt notational styles to different communities (the controls community will always insist that the state is x_t while decisions (controls) are u_t), we believe that this article provides a path to helping communities communicate using a common framework centered on the search for policies (which are functions) rather than deterministic variables.

References

[1] R. Agrawal. The continuum-armed bandit problem. *SIAM Journal on Control and Optimization*, 33(6):19–26, 1995.

[2] Peter Auer, N Cesa-bianchi, and P Fischer. Finite time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256, 2002.

[3] Emre Barut and Warren B. Powell. Optimal learning for sequential sampling with non-parametric beliefs. *J. Global Optimization*, 58:517–543, 2014.

[4] Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. *Robust Optimization*. Princeton University Press, Princeton NJ, 2009.

[5] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. II: Approximate Dynamic Programming*. Athena Scientific, Belmont, MA, 4 edition, 2012.

[6] J. R. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer, New York, 2nd edition, 2011.

[7] Ronen I. Brafman and Moshe Tennenholtz. R-max A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning. *Journal of Machine Learning Research*, 3(2):213–231, feb 2003.

[8] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Trans. on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[9] Sébastien Bubeck and Nicolò Cesa-Bianchi. Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems. *Foundations and Trends® in Machine Learning*, 5(1):1–122, 2012.

[10] Sébastien Bubeck, Rémi Munos, Gilles Stoltz, and Csaba Szepesvári. X -Armed Bandits. *Journal of Machine Learning Research*, 12:1655–1695, 2011.

[11] Alexandra Carpentier. Bandit Theory meets Compressed Sensing for high-dimensional Stochastic Linear Bandit. *Aistats*, XX:190–198, 2012.

[12] Chun-Hung Chen and Loo Hay Lee. *Stochastic Simulation Optimization*. World Scientific Publishing Co., Hackensack, N.J., 2011.

[13] Si Chen, Kristofer-Roy G Reyes, Maneesh Gupta, Michael C Mcalpine, and Warren B. Powell. Optimal learning in Experimental Design Using the Knowledge Gradient Policy with Application to Characterizing Nanoemulsion Stability. *SIAM/ASA J. Uncertainty Quantification*, 3:320–345, 2015.

[14] Erhan Cinlar. *Introduction to Stochastic Processes*. Prentice Hall, Upper Saddle River, NJ, 1975.

[15] Andrew R. Conn, Katya Scheinberg, and Luis N. Vicente. *Introduction to Derivative-Free Optimization*. SIAM Series on Optimization, Philadelphia, 2009.

[16] Adrien Couetoux, Jean-Baptiste Hoock, Nataliya Sokolovska, Olivier Teytaud, and Nicolas Bonnard. Continuous Upper Confidence Trees. pages 433–445. Springer Berlin Heidelberg, 2011.

[17] George B Dantzig. Linear programming with uncertainty. *Management Science*, 1:197–206, 1955.

[18] Savas Dayanik, Warren B. Powell, and Kazutoshi Yamazaki. Asymptotically optimal Bayesian sequential change detection and identification rules. *Annals of Operations Research*, 208(1):337–370, apr 2012.

[19] M H DeGroot. *Optimal Statistical Decisions*. John Wiley and Sons, 1970.

[20] Marc Peter Deisenroth, Gerhard Neumann, and Jan Peters. A Survey on Policy Search for Robotics. *Foundations and Trends in Machine Learning*, 2(1-2):1–142, 2011.

[21] Peter I. Frazier, Warren B. Powell, and S. Dayanik. The Knowledge-Gradient Policy for Correlated Normal Beliefs. *INFORMS Journal on Computing*, 21(4):599–613, may 2009.

[22] Peter I. Frazier, Warren B. Powell, and S. E. Dayanik. A knowledge-gradient policy for sequential information collection. *SIAM Journal on Control and Optimization*, 47(5):2410–2439, 2008.

[23] Michael C. Fu. *Handbook of Simulation Optimization*. Springer, New York, 2014.

[24] Abraham P. George and Warren B. Powell. Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming. *Journal of Machine Learning*, 65(1):167–198, 2006.

[25] J Ginebra and M K Clayton. Response Surface Bandits. *Journal of the Royal Statistical Society. Series B (Methodological)*, 57:771–784, 1995.

[26] J.C. Gittins, Kevin D. Glazebrook, and R. R. Weber. *Multi-Armed Bandit Allocation Indices*. John Wiley & Sons, New York, 2011.

[27] J.C. Gittins and D.M. Jones. A dynamic allocation index for the sequential design of experiments. In J. Gani, editor, *Progress in statistics*, pages 241—-266. North Holland, Amsterdam, 1974.

[28] T Hastie, R Tibshirani, and J Friedman. *The elements of statistical learning: data mining, inference and prediction*. Springer, New York, 2009.

[29] Kenneth L. Judd. *Numerical Methods in Economics*. MIT Press, 1998.

[30] L. P. Kaelbling. *Learning in embedded systems*. MIT Press, Cambridge, MA, 1993.

[31] Peter Kall and Stein W Wallace. *Stochastic Programming*. 2003.

[32] M. Kearns and Satinder P Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning, 49*, 2(3):209–232, 2002.

[33] A. J. Kleywegt, Alexander Shapiro, and Tito Homem-de Mello. The sample average approximation method for stochastic discrete optimization. *SIAM J. Optimization*, 12(2):479–502, 2002.

[34] T L Lai and Herbert Robbins. Asymptotically Efficient Adaptive Allocation Rules. *Adv. Appl. Math.*, 6:4–22, 1985.

[35] Frank L. Lewis, D.L. Vrabie, and V. L Syrmos. *Optimal Control*. John Wiley & Sons, Hoboken, NJ, 3rd edition, 2012.

[36] Keqin Liu and Qing Zhao. Indexability of restless bandit problems and optimality of Whittle index for dynamic multichannel access. *IEEE Transactions on Information Theory*, 56(11):5547–5567, 2010.

[37] Matthew S. Maxwell, Shane G Henderson, and Huseyin Topaloglu. Tuning approximate dynamic programming policies for ambulance redeployment via direct search. *Stochastic Systems*, 3(2):322–361, 2013.

[38] Martijn R K Mes, Warren B. Powell, and Peter I. Frazier. Hierarchical Knowledge Gradient for Sequential Sampling. *Journal of Machine Learning Research*, 12:2931–2974, 2011.

[39] R Munos and Csaba Szepesvári. Finite time bounds for fitted value iteration. 1:815–857, 2008.

[40] J. M. Nascimento and Warren B. Powell. An Optimal Approximate Dynamic Programming Algorithm for Concave, Scalar Storage Problems With Vector-Valued Controls. *IEEE Transactions on Automatic Control*, 58(12):2995–3010, dec 2013.

[41] D. M. Negoescu, Peter I. Frazier, and Warren B. Powell. The Knowledge-Gradient Algorithm for Sequencing Experiments in Drug Discovery. *INFORMS Journal on Computing*, 23(3):346–363, dec 2011.

[42] Arkadi Nemirovski, a Nemirovski, a Juditsky, G Lan, and Alexander Shapiro. *Robust stochastic approximation approach to stochastic programming* , volume 19. 2009.

[43] José Niño-Mora. Computing a classic index for finite-horizon bandits. *INFORMS Journal on Computing*, 23(2):254–267, 2011.

[44] M.V. F. Pereira and L. M. V. G. Pinto. Multi-stage stochastic optimization applied to energy planning. *Mathematical Programming*, 52:359–375, 1991.

[45] Warren B. Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*. John Wiley & Sons, Hoboken, NJ, 2 edition, 2011.

[46] Warren B. Powell. Clearing the Jungle of Stochastic Optimization. *Informs TutORials in Operations Research 2014*, (October), 2014.

[47] Warren B. Powell and Ilya O. Ryzhov. *Optimal Learning*. John Wiley & Sons, Hoboken, NJ, 2012.

[48] M. Puterman. *Markov Decision Processes*. John Wiley & Sons Inc, Hoboken, NJ, 2nd edition, 2005.

[49] Herbert Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

[50] R T Rockafellar and R J.-B. Wets. Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of Operations Research*, 16(1):119–147, 1991.

[51] Ilya O. Ryzhov and Warren B. Powell. Bayesian active learning with basis functions. *IEEE SSCI 2011: Symposium Series on Computational Intelligence - ADPRL 2011: 2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, (3):143–150, 2011.

[52] Ilya O. Ryzhov, Warren B. Powell, and Peter I. Frazier. The Knowledge Gradient Algorithm for a General Class of Online Learning Problems. *Operations Research*, 60(1):180–195, mar 2012.

[53] Alexander Shapiro. Analysis of stochastic dual dynamic programming method. *European Journal of Operational Research*, 209(1):63–72, feb 2011.

[54] Alexander Shapiro, D. Dentcheva, and Andrzej Ruszczyński. *Lectures on Stochastic Programming: Modeling and theory*. SIAM, Philadelphia, 2 edition, 2014.

[55] Jennie Si, Andrew G Barto, Warren B. Powell, and D. Wunsch. Handbook of Learning and Approximate Dynamic Programming. *Wiley-IEEE Press*, 2004.

[56] Hugo P. Simao, J Day, Abraham P. George, T. Gifford, Warren B. Powell, and J Nienow. An Approximate Dynamic Programming Algorithm for Large-Scale Fleet Management: A Case Application. *Transportation Science*, 43(2):178–197, 2009.

[57] Marta Soare, Alessandro Lazaric, and Rémi Munos. Best-Arm Identification in Linear Bandits. *Proceedings of the 28th Annual Conference on Neural Information Processing Systems (NIPS'14)*, pages 1–17, 2014.

[58] James Spall. *Introduction to Stochastic Search and Optimization: Estimation, simulation and control*. John Wiley & Sons, Hoboken, NJ, 2003.

[59] Richard S. Sutton and Andrew G Barto. *Reinforcement Learning*, volume 35. MIT Press, Cambridge, MA, 1998.

[60] Csaba Szepesvári. *Algorithms for Reinforcement Learning*, volume 4. Morgan and Claypool, jan 2010.

[61] Yingfei Wang, Warren B. Powell, and Robert Schapire. Finite-time analysis for the knowledge-gradient policy and a new testing environment for optimal learning. Technical report, Princeton University, Princeton, N.J., 2015.

[62] Peter Whittle. Sequential Decision Processes with Essential Unobservables. *Advances in applied mathematics(Print)*, 1(2):271–287, 1969.