CHAPTER 18

OPTIMAL LEARNING AND APPROXIMATE DYNAMIC PROGRAMMING

Warren B. Powell and Ilya O. Ryzhov

Princeton University, University of Maryland

18.1 INTRODUCTION

Approximate dynamic programming (ADP) has emerged as a powerful tool for tackling a diverse collection of stochastic optimization problems. Reflecting the wide diversity of problems, ADP (including research under names such as reinforcement learning, adaptive dynamic programming and neuro-dynamic programming) has become an umbrella for a wide range of algorithmic strategies. Most of these involve learning functions of some form using Monte Carlo sampling.

A recurring theme in these algorithms involves the need to not just learn policies, but to learn them quickly and effectively. Learning arises in both offline settings (training an algorithm within the computer) and online settings (where we have to learn as we go). Learning also arises in different ways within algorithms, including learning the parameters of a policy, learning a value function and learning how to expand the branches of a tree.

The focus of this chapter is to discuss different strategies for learning policies in an efficient way. These "strategies" are themselves referred to as policies, as they represent rules for guiding the collection of information. Passive learning policies will collect information and update beliefs about functions, without making any explicit attempt at collecting information in a way that would accelerate the learning policy. Active learning refers to policies where we are willing to make suboptimal actions explicitly because the information gained will add value later in the process. Most of the literature has focused on simple heuristics, but in some cases these heuristics have provable suboptimality bounds. Our presentation will focus primarily on the knowledge gradient policy, which maximizes the rate of learning, and offers both theoretical and practical features in a Bayesian setting which focuses on minimizing expected opportunity cost.

To describe the different ways in which active learning arises, we need to first understand different problem settings and, most importantly, the major classes of policies. Section 18.2 provides a compact modeling framework and highlights some important problem classes. After this, Section 18.3 identifies the major classes of policies, which we then use to identify opportunities for optimal learning.

18.2 MODELING

There are five core components of any stochastic, dynamic system. These include

- The state variable S_t - The state variable captures the information available at time t.
- Decisions, actions or controls - We use the notation a_t to represent discrete action spaces, although many problems exhibit vector-valued decisions or controls.
- Exogenous information - We let W_t represent any random information that becomes known for the first time at discrete time t.
- The transition function - Also known as the system model, plant model or just model, we represent it using the function $S^M(\cdot)$ where we will write

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

  to represent the evolution of the state variable over time due to the effect of an action a_t and new information W_{t+1}.
- Objective function - We let $C(S_t, a_t)$ be the contribution (if maximizing) or cost (if minimizing) that depends on the state S_t and action a_t. In some applications, the one-period contribution may be random at time t, in which case we would write it as $C(S_t, a_t, W_{t+1})$. In this case, we assume that $C(S_t, a_t)$ is the expected contribution.

For our purposes, we need to further identify three perspectives of a state variable:

- The physical state - Sometimes called the resource state, there are many problems where we are managing a resource such as robots, a fleet of vehicles, a chemical process or financial investments. The resource state, R_t, captures the state of physical entities that are being managed.

- The information state - Widely used in control theory, we let I_t capture information used to make a decision, which would include R_t but may also include information such as prices and the weather.

- The knowledge (or belief) state - The knowledge state captures our belief about unobservable parameters. This is a particularly important dimension of the problems we consider in this chapter, and represents one of the more subtle and difficult dimensions of dynamic programming.

For the moment, we do not address the problem of how we make decisions, other than to assume there is some rule, or *policy* for making decisions. The research literature will typically represent a policy as π. We prefer the notation that if we are determining an action a, our policy would be written as a function $A^\pi(S_t)$ where π parameterizes both the class of function (examples are given below), and any parameters that determine the behavior of the function. If we use decision x, our decision function would be written $X^\pi(S_t)$.

We need an objective function that defines the best policy. We assume that we are maximizing expected total (possibly discounted) contributions, recognizing that there are other objective functions. The problem of finding the best policy can be written

$$\max_\pi \mathbb{E}^\pi \sum_{t=0}^{T} \gamma^t C(S_t, A^\pi(S_t)). \tag{18.1}$$

Equation (18.1) is the formal definition of a dynamic program, which is a sequential decision process. We are generally not able to find a policy that solves (18.1), so we have to resort to finding an approximate policy. This is the essence of approximate dynamic programming.

18.3 THE FOUR CLASSES OF POLICIES

The literature on stochastic optimization can sometimes be viewed as a jungle of modeling and algorithmic strategies. Terms like "policy search," "reinforcement learning," "stochastic programming," "simulation-optimization," "model predictive control," "Monte Carlo tree search" and "actor-critic methods" (to name a few) are often used without a clear definition.

Without question, perhaps one of the most confusing and misunderstood terms in dynamic programming is the word *policy*. Stated simply, a policy is a mapping from state to action. These mappings are typically written as functions, and are sometimes confusingly described as rules.

The research community has created a complex array of functions for making decisions. These can generally be boiled down to four fundamental classes, with many opportunities for creating hybrid policies. We list these because the structure of each policy introduces specific learning challenges.

18.3.1 Myopic cost function approximation

There is the rare problem that lends itself to a simple myopic policy of the form

$$A^M(S_t) = \arg\max_a C(S_t, a).$$

One example involves balancing financial portfolios when there are no transaction costs. We can set up a model where we maximize one-period returns plus a penalty for the aggregate volatility of the account. Without transaction costs, we can move funds from one asset to another with no cost. As a result, a policy that optimizes the one-period utility is optimal.

Myopic policies are often used as heuristics in specific applications. We might have a resource i (such as a truck) that we are looking to assign to task j (such as a load). Let c_{ij} be the cost of assigning truck i to load j. A myopic policy would be to solve

$$A^M(S_t) = \arg\min_a \sum_i \sum_j c_{ij} a_{ij},$$

subject to

$$\begin{aligned} \sum_i a_{ij} &\leq 1, \\ \sum_j a_{ij} &\leq 1, \\ a_{ij} &\geq 0. \end{aligned}$$

We further assume that unassigned drivers and loads are held to the next time period. This policy is actually used in some commercial dispatch systems in the truckload trucking industry, but it is hardly optimal. For example, it ignores how long a load has been held. A proper dynamic program would capture how long drivers and loads have been sitting, and would make a decision that balances minimizing costs and service delays over an entire horizon. Since this is hard to solve, a simple heuristic is to introduce a bonus for moving loads that have been waiting. Letting τ_i be the length of time that a load has been waiting, the policy would be given by

$$A^M(S_t|\theta) = \arg\min_a \sum_i \sum_j (c_{ij} - \theta\tau_i) a_{ij}. \tag{18.2}$$

Here we have created a family of myopic policies parameterized by θ.

18.3.2 Lookahead policies

Lookahead policies come in many forms, but they all involve optimizing explicitly over multiple time periods, but then just retaining the decision that we implement now. The most common type of lookahead uses a point forecast of any random information, which transforms the model into a deterministic optimization problem of the form

$$A^L(S_t) = \arg\max_t \left(c_t a_t + \sum_{t'=t+1}^{t+\bar{t}} c_{t'} a_{t'} \right).$$

Here, we define $\arg\max_t$ as the operator that maximizes over the vector $a_t, \ldots, a_{t+\bar{t}}$, but only returns a_t. This strategy is often referred to as a rolling horizon procedure, because we optimize over a horizon of length $\bar{T}$, implement the decision for time t, and then roll forward, updating any parameters due to new information.

There are several communities that work to incorporate uncertainty into the lookahead process, such as decision trees (for problems with small action spaces) or stochastic programs (for vector-valued decisions; see Birge and Louveaux, 1997). Monte Carlo tree search (MCTS) has become a popular field of research in the reinforcement learning community, contributing to breakthroughs on difficult problems such as computer Go (Silver, 2009; Gelly and Wang, 2006; Coulom, 2006).

18.3.3 Policy function approximation

We use the term "policy function approximation" to refer to direct mappings from states to actions, without the need to solve an embedded optimization problem. Examples of policy function approximations include

- If a patient has a set of symptoms s, choose drug d (for every state, we have to have an associated drug).
- If the inventory in a store is less than q, order enough to bring it up to Q.
- If the reservoir in a lake is of level h, set the rate of outflow from the dam to $x = \theta_0 + \theta_1 h$.
- There is a set of discrete actions a with contribution $C(s, a)$ when we are in state s. We are going to choose action a with probability

  $$p(a, s|\theta) = \frac{e^{\theta C(s,a)}}{\sum_{a'} e^{\theta C(s,a)}}$$

  where θ is a tunable parameter. As θ is increased, we increase the likelihood that we will choose action a.

In the drug example, we have a parameter (the choice of drug d) for each state. Let θ_s be the choice of drug if we are in state s. In the inventory example, we might write $\theta = (q, Q)$. In the lake example, the decision is determined by a function parameterized by $\theta = (\theta_0, \theta_1)$. Note that while the second and third policies have only two parameters, the drug policy has a parameter for every state. The fourth policy (which is widely used in reinforcement learning) is parameterized by a scalar θ.

18.3.4 Policies based on value function approximations

It is well known that we can mathematically characterize an optimal policy using Bellman's optimality equation which we write using

$$\begin{aligned} V_t(S_t) &= \max_a \left(C(S_t, a) + \gamma \sum_{s'} p(s'|S_t, a) V_{t+1}(s') \right) && (18.3) \\ &= \max_a \left(C(S_t, a) + \gamma \mathbb{E}\{V_{t+1}(S^M(S_t, a, W_{t+1}))|S_t\} \right). && (18.4) \end{aligned}$$

We use the time dependent version because it makes the modeling clearer (in addition, there are many problems where policies are time dependent). Equation (18.3) is the form most widely used in texts on Markov decision processes (such as Puterman, 2005), while equation (18.4) is known as the expectation form, where the expectation is over the random variable W_{t+1} given that we are in state S_t.

There is an rich literature that has evolved around the idea of using Bellman's equation to derive optimal (or at least good) policies. We face several challenges when going down this path. First, for most applications we cannot compute the expectation. We circumvent this by using the concept of the post-decision state S_t^a, which is the state after making a decision but before any new information has arrived. Second, we cannot compute the value function exactly, we replace it with a statistical approximation. While we can tap a wide array of approximation strategies (see, for example, Hastie et al., 2009), the most popular strategy is a simple linear model of the form

$$\bar{V}_t(S_t^a|\theta_t) = \sum_{f \in \mathcal{F}} \theta_{tf} \phi_f(S_t^a). \qquad (18.5)$$

where $(\phi_f(S_t^a))_{f \in \mathcal{F}}$ is a vector of user-specified *basis functions* and $(\theta_f)_{f \in \mathcal{F}}$ is a vector of regression parameters. This allows us to write our policy as

$$A_t^{\pi,n}(S_t|\theta_t) = \arg\max_a \left(C(S_t^n, a) + \gamma \sum_{f \in \mathcal{F}} \theta_{tf}^n \phi_f((S^a(S_t^n, a))) \right). \qquad (18.6)$$

Despite the elegance and simplicity of this general strategy, it requires decisions about a range of algorithmic choices. Provably convergent algorithms are rare and generally require fairly strong assumptions (for example, the policy may be fixed rather than optimized; see Sutton et al., 2009, for a recent example). There are different strategies for generating sample estimates of the value of being in a state, estimating the value of a policy versus updating a policy, applying on-policy versus off-policy learning, and state sampling. One issue that arises is that the information we collect to update the value function approximation depends on what state we visit, and we have some control over the state.

For in-depth discussions of the use of value function approximations, we refer the reader to Chapter 6 in Bertsekas (2011a), Bertsekas (2011b), Bertsekas and Tsitsiklis (1996), Sutton and Barto (1998), Szepesvari (2010) and Chapters 8-10 of Powell (2011), and the many references cited there.

18.3.5 Learning policies

We have outlined above four fundamental classes of policies. Needless to say, it is possible to create a host of additional policies by mixing and matching these strategies. One popular strategy is to use a limited tree search with a value function approximation at the end of the tree. Cost function approximations are widely used to improve myopic policies, but just as often will be merged with a rolling horizon procedure (again, a form of tree search).

Three of these fundamental policies involve some sort of functional approximation: approximating a cost function, approximating a function that specifies the policy, and approximating a value function which in turn is used to define a policy. There are three ways of approximating functions: lookup tables, parametric models and nonparametric models. With a lookup table, specifying a discrete state returns an action (policy function approximation) or the value of being in a state (value function approximation). Parametric models, used for all three types of functional approximations, represent any analytical function with a predetermined set of parameters to be determined. Nonparametric models have also been suggested for both policies (primarily for engineering applications with continuous actions) and value functions.

Special challenges arise when we have to learn value function approximations. There is a major difference between learning the parameters θ that govern the behavior of a policy, and learning value functions. As we show below, the first can be modeled as a dynamic program with a pure belief state, while the second requires that we handle the presence of a physical state.

Below, Section 18.4 provides an introduction to the basic problem of tuning the parameters of a policy and introduces several popular heuristic policies. Then, Section 18.5 presents optimal policies for learning, including a characterization of the optimal policy for learning as a dynamic program with a pure belief state. Finally, Section 18.6 closes with a discussion of optimal learning in the presence of a physical state, which is the challenge we face in approximate dynamic programming.

18.4 BASIC LEARNING POLICIES FOR POLICY SEARCH

All four classes of policies above can be represented as a function $A^\pi(S_t|\theta)$ if we wish to choose a discrete action a_t. The index π specifies the class of function, while θ captures all the tunable parameters that determine the behavior of the function within the class of policy (we should write the tunable parameter vector as θ^π to express its dependence on the class of policy, but we suppress this bit of additional notation for simplicity). We emphasize that $A^\pi(S_t|\theta)$ may be *any* of the four fundamental classes of policies, including policies based on value function approximations.

Once we have chosen the class of policy π, the problem of finding the best vector θ requires solving

$$\min_\theta F^\pi(\theta) = \mathbb{E}\sum_{t=0}^{T} C(S_t, A^\pi(S_t|\theta)). \tag{18.7}$$

Equation (18.7) is a classical problem in stochastic optimization that was first addressed in the seminal paper by Robbins and Monro (1951). Recognizing that the expectation cannot be computed for any but the most specialized problems, we use our ability to simulate a policy, or to observe a policy being used in a real situation. Given a sample path ω, let $F^\pi(\theta, \omega)$ be a simulation of the value of policy π. If we further assume that we can compute the gradient $\nabla_\theta F^\pi(\theta, \omega)$ (along with some other assumptions), the classic stochastic approximation procedure would allow us to find θ using

$$\theta^n = \theta^{n-1} - \alpha_{n-1}\nabla_\theta F^\pi(\theta^{n-1}, \omega^n). \tag{18.8}$$

Equation (18.8) is a simple, elegant optimization algorithm which offers provable asymptotic convergence (for reviews, see Chang et al., 2007; Fu, 2008). At the same time, there are no promises on rate of convergence: while this algorithm can work quite well in practice, it can be slow. Finally, there are many problems where we cannot find the gradient. There are hundreds of papers addressing the many variations of this fundamental problem. For reviews, see Spall (2003), Ruszczynski and Shapiro (2003) and Shapiro et al. (2009).

Our focus is not just learning the best policy, but learning it quickly, with the ultimate (if unachievable) goal of learning it at an optimal rate. Rate of convergence is particularly useful when computing a sample realization of $F^\pi(\theta^{n-1}, \omega^n)$ is slow. It is not hard to find industrial models where calculating $F^\pi(\theta^{n-1}, \omega^n)$ can take anywhere from 30 minutes to a day or more. There are settings where $F^\pi(\theta^{n-1}, \omega^n)$ is not a computer simulation but rather is a physical system. We might be interested in a sales policy where we would fix θ and then observe sales for a week or more.

The literature on optimal learning typically focuses on solving a problem of the form

$$\min_x \mathbb{E}F(x, W), \tag{18.9}$$

where we have to find the best value $x \in \mathcal{X}$ within a finite measurement budget. In our setting, it is more natural to use θ to parameterize the policy (in some communities, x is used as the decision variable within the dynamic program). It should be apparent that equations (18.7) and (18.9) are equivalent. We are going to let θ_x be a particular value of θ corresponding to $x \in \mathcal{X}$, so that there is a one-to-one relationship between x and θ, allowing us to bridge two notational systems.

In practice x (or θ_x) may be a) a small, discrete set, b) a large, discrete set (that is, too large to enumerate), c) a continuous scalar, or d) a vector of any form. We do not have the space to describe specialized algorithms for all these cases, but we are going to cover algorithms that can handle a wide range of applications that arise in practice, including problems where x is continuous.

18.4.1 The belief model

We proceed by first forming a belief model about $F(x) = F^\pi(\theta_x)$. If $x \in \{1, 2, \ldots, M\}$, it is standard practice to use a Bayesian model where μ_x is the unknown true value of $F(x)$ where $x = \theta_x$. We treat μ_x as a random variable which is normally distributed with mean μ_x^0 and precision $\beta_x^0 = 1/(\sigma_x^0)^2$, where the precision is the inverse of the variance. We refer to (μ_x^0, β_x^0) as our prior (in addition to the assumption of normality). We have to specify these parameters based on some domain knowledge (or preliminary simulations). They reflect our belief about the likely values of μ_x.

We are going to start by assuming that we have a lookup table representation of $F(\theta)$, which is to say that there is a true value $\mu_x = \mathbb{E}F(\theta_x)$ giving us how well the policy performs if $\theta = \theta_x$. For our lookup table representation, we assume that we are searching for the best value of θ among a finite set of vectors $(\theta_x)_{x \in \mathcal{X}}$, $\mathcal{X} = \{1, 2, \ldots, M\}$. We let (μ_x^n, β_x^n) be our belief about $F(\theta_x)$ after n observations. For a Bayesian model with a normal prior and normally distributed observations W_x^n of $F(x)$, beliefs are updated using

$$\mu_x^{n+1} = \frac{\beta_x^n \mu_x^n + \beta^W W_x^{n+1}}{\beta_x^n + \beta^W}, \tag{18.10}$$

$$\beta_x^{n+1} = \beta_x^n + \beta^W. \tag{18.11}$$

Here, $\beta^W = 1/\sigma_W^2$ is the precision (inverse variance) of a measurement W_x^n. We assume that β^W is independent of x, but this is easily generalized.

18.4.2 Objective functions for offline and online learning

Assume that we have a budget of N measurements to find the best value of x. Let μ_x^N be our belief about $F(x) = F(\theta_x)$ after N measurements which were chosen according to policy π, and let

$$x^\pi = \arg\max_x \mu_x^N$$

be our choice of the optimal solution while using policy π. This problem is widely known as the ranking and selection problem but we prefer the term offline learning, where we do not care about rewards earned while we are trying to learn the best choice.

Our goal is to find a policy π that solves the following maximization problem

$$\max_\pi \mathbb{E}_\mu \mathbb{E}_W \mu_{x^\pi}. \tag{18.12}$$

The objective function in (18.12) reflects the fact that there are two sets of random variables we have to deal with. First, the true values μ_x are normally distributed with mean μ^0 and precision β^0. Second, the observations $W^1, \ldots, W^N$ are observed from distributions determined by μ and our policy π, and will affect the outcome of x^π.

If we are learning as we progress (known as online learning), our objective function becomes

$$\max_\pi \mathbb{E}_\mu \mathbb{E}_W \sum_{n=1}^{N} \gamma^n \mu_{x^n}, \tag{18.13}$$

where $x^n = \arg\max_{x'} \mu_{x'}^{n-1}$ and $0 < \gamma \leq 1$ is a discount factor. Online learning is widely referred to as the multi-armed bandit problem in the research literature (see Gittins et al., 2011, for a thorough review). It has drawn significant attention in the applied probability and computer science communities. If we are tuning a policy, most of the time we are doing this in an offline fashion within a computer, but we may have to design policies in real time, incurring costs and rewards as we are learning the policy.

18.4.3 Some heuristic policies

Some of the best known heuristic policies include:

- Pure exploitation - This policy uses

  $$x^n = \arg\max_{x \in \mathcal{X}} \mu_x^n.$$

  Pure exploitation makes the most sense for online learning problems, where there is value in getting the best solution at each iteration. However, this is widely used in response-surface methods, where at each iteration we update a parametric approximation of $F(x)$.

- Pure exploration - This policies chooses x at random. This approach makes sense in off-line settings, where we are using observations to fit a parametric approximation of $F(x)$ which is then optimized to find the best value of x.

- Epsilon-greedy exploration - This is a hybrid, where we choose x at random (explore) with probability ϵ, and choose what appears to be the best (exploit) with probability $1 - \epsilon$. Typically ϵ declines with the number of iterations. Epsilon-greedy focuses more energy on evaluating the option that appears to be the best, but still provides for a level of exploration in case we are missing what is truly the best option. See Sutton and Barto (1998) for an introduction to epsilon-greedy and Singh et al. (2000) for an analysis of convergence properties.

- Boltzmann exploration - Here we choose x with probability

  $$p_x^n = \frac{\exp(\rho \mu_x^n)}{\sum_{x' \in \mathcal{X}} \exp(\rho \mu_{x'}^n)}.$$

This policy overcomes a major limitation of epsilon-greedy, which does not treat the second best option any differently than the worst. If $\rho = 0$, Boltzmann-exploration is equivalent to pure exploration, while as ρ increases, it approaches pure exploitation. Boltzmann exploration is very popular in the reinforcement learning community as a parameterized random policy, where it is sometimes referred to as a soft-max policy.

- Interval estimation - For each x, we compute

$$\nu_x^{IE,n} = \mu_x^n + z_\alpha \sigma_x^n$$

where z_α is a tunable parameter (often set to around 2 or 3), and where $(\sigma_x^n)^2 = 1/\beta_x^n$ is the variance of our beliefs about μ_x. IE was introduced by Kaelbling (1993).

- Upper confidence bounding - A UCB policy, designed specifically for normally distributed measurements, is calculated using

$$\nu_x^{UCB,n} = \mu_x^n + 4\sigma_W \sqrt{\frac{\log n}{N_x^n}}. \quad (18.14)$$

Here, N_x^n is the number of times we have observed x through time n.

Pure exploration and exploitation are heuristics with obvious weaknesses, but exploration is often used in offline learning where you observe the function at random locations, fit a function and then choose the best. By contrast, pure exploitation is widely used in practice for online learning problems, where choosing what appears to be the best is an understandable if imperfect policy. Epsilon-greedy and Boltzmann exploration are both hybrid exploration-exploitation policies.

Interval estimation is the first policy that uses both the current estimate of our belief μ_x^n about $F(x)$, and the uncertainty in this belief. Here, n refers to the number of times we have observed x. It is important to tune the parameter z_α, but properly tuned, IE has been found to work well in practice.

Upper confidence bounding, known generally as UCB policies, is similar to interval estimation in that there is a term added to the estimate μ_x^n that captures how good μ_x *might* be. UCB policies have attracted considerable interest because it is possible to prove that there is a logarithmic bound in N on suboptimality (or *regret*) in online learning. Regret bounds on UCB policies were first introduced by Lai and Robbins (1985); the UCB policy presented here is given in Auer et al. (2002). Researchers have started to develop similar policies for offline learning problems with similar regret bounds, but as of this writing, experimental evaluations of these policies are limited (see Bubeck et al., 2009, for an example).

We note that classical stochastic gradient methods such as the one in equation (18.8) do not use any form of learning. After observing a stochastic gradient, we update our estimate of the best value of θ, but we do not otherwise update any belief models. By contrast, we assume that when using any of the policies listed in this section, we would use equations (18.10)-(18.11) to update our beliefs about the choice we just observed. Pure exploitation makes no effort to collect information purely for the value of learning, while all the remaining policies use some form of explicit exploration, where we are willing to observe a value x that we do not believe is best, because it is possible that it *might* be best. In our own experiments, interval estimation seems to work extremely well if z_α is properly tuned (and this tuning can have a real impact on performance). By contrast, we have had less success with UCB policies, but these policies are supported by rigorous bounds. However, these bounds also grow with N, so empirical performance may vary.

18.5 OPTIMAL LEARNING POLICIES FOR POLICY SEARCH

Ideally, we would have a way to find the policy π that produces the best possible result, whether it is offline or online. It turns out we can characterize the optimal policy by formulating the problem as a dynamic program, first presented in DeGroot (1970). Assume that everything is still normally distributed. Then, our state variable is our state of belief, which is given by $S^n = (\mu_x^n, \beta_x^n)_x$. Bellman's equation then characterizes the optimal policy using

$$V^n(S^n) = \max_x \left(C(S^n, x) + \gamma \mathbb{E}\{V^{n+1}(S^{n+1}(s, x))|S^n\}\right), \tag{18.15}$$

where S^{n+1} is given by equations (18.10) and (18.11). Unfortunately, (18.15) is computationally intractable. It is important to emphasize that the problem setting of learning the parameters of a policy can be formulated as a dynamic program with just a belief state. Later, we consider the much more difficult problem of combining a belief state with a physical state.

For this basic model, the optimal policy can be characterized using Gittins indices, given by

$$\nu_x^{Gitt,n}\left(\mu_x^n, \sigma_x^n, \sigma_W, \gamma\right) = \mu_x^n + \sigma_W G\left(\frac{\sigma_x^n}{\sigma_W}, \gamma\right). \tag{18.16}$$

This theory, described in depth in Gittins et al. (2011), is very elegant, but the functions $G\left(\frac{\sigma_x^n}{\sigma_W}, \gamma\right)$, known as Gittins indices, are hard to compute. Chick and Gans (2009) has developed a simple analytical approximation to overcome this problem, but Gittins indices are only optimal within a relatively simple class of problems that does not arise often in practice. As an alternative, we propose the use of the knowledge gradient, a simple concept that can be extended to handle more general settings.

18.5.1 The knowledge gradient for offline learning

Let

$$V^n(S^n) = \max_{x'} \mu_{x'}^n$$

be the current value of our problem given the state of knowledge S^n after n iterations. Then let $S^{n+1}(x)$ be the belief state if we choose to measure $x^n = x$ and observe

W_x^{n+1} (which is still random since we have not actually performed the observation), which means that $V^{n+1}(S^{n+1}(x))$ is a random variable. The knowledge gradient is a policy where we choose to measure the alternative x that gives us the greatest value of information, and is given by

$$\nu_x^{KG,n} = \mathbb{E}\left[V^{n+1}(S^{n+1}(x)) - V^n(S^n)|S^n\right]. \tag{18.17}$$

Here, S^n is our current state of knowledge (that is, the beliefs about all the alternatives), and $S^{n+1}(x)$ is the state of knowledge if we choose to observe x, but before we have actually made the observation. The concept is simple; the trick is computing it.

Up to now, we have focused on problems with independent beliefs, which is to say that if we choose to measure x and observe W_x^n, we do not learn anything about alternatives $x' \neq x$. For this problem, computing the knowledge gradient is quite simple. We first have to find the conditional *change* in the variance of an estimate given a measurement, given by

$$\tilde{\sigma}_x^{2,n} = Var^n[\mu_x^{n+1} - \mu_x^n].$$

It is possible to show that this is given by

$$\begin{aligned} \tilde{\sigma}_x^{2,n} &= \sigma_x^{2,n} - \sigma_x^{2,n+1} && (18.18)\\ &= (\beta_x^n)^{-1} - (\beta_x^n + \beta^W)^{-1}. && (18.19) \end{aligned}$$

We next compute ζ_x^n which is given by

$$\zeta_x^n = -\left|\frac{\mu_x^n - \max_{x' \neq x} \mu_{x'}^n}{\tilde{\sigma}_x^n}\right|. \tag{18.20}$$

Now let

$$f(\zeta) = \zeta\Phi(\zeta) + \phi(\zeta), \tag{18.21}$$

where $\Phi(\zeta)$ and $\phi(\zeta)$ are, respectively, the cumulative standard normal distribution and the standard normal density.

We can now express the knowledge gradient as

$$\nu_x^{KG,n} = \tilde{\sigma}_x^n f(\zeta_x^n). \tag{18.22}$$

The knowledge gradient policy chooses to measure the value x which maximizes $\nu_x^{KG,n}$ over all x.

Compared to solving the dynamic program in equation (18.15), the knowledge gradient is simply a one-step lookahead policy. However, in contrast with problems with a physical state, this policy seems to work extremely well for pure learning problems. For offline learning problems, it offers several nice theoretical properties:

1) The knowledge gradient is always positive, $\nu_x^{KG,n} \geq 0$ for all x. Thus, if the knowledge gradient of an alternative is zero, that means we won't measure it.

2) The knowledge gradient policy is optimal (by construction) if we are going to make exactly one measurement.

3) If there are only two choices, the knowledge gradient policy is optimal for any measurement budget N.

4) If N is our measurement budget, the knowledge gradient policy is guaranteed to find the best alternative as N is allowed to be big enough. That is, if x^N is the solution we obtain after N measurements, and

$$x^* = \arg\max \mu_x$$

is the true best alternative, then $x^N \to x^*$ as $N \to \infty$. This property is known as asymptotic optimality or consistency.

5) There are many heuristic policies that are asymptotically optimal (for example, pure exploration, mixed exploration-exploitation, epsilon-greedy exploration and Boltzmann exploration). But none of these heuristic policies are myopically optimal. The knowledge gradient policy is the only pure policy (a more precise term would be to say it is the only stationary policy) that is both myopically and asymptotically optimal.

6) The knowledge gradient has no tunable parameters.

Myopic optimality and asymptotic optimality are the two properties that suggest that the knowledge gradient will have good convergence over intermediate budgets.

The one issue that requires some caution is known as the S-curve effect, that arises when the marginal value of information is nonconcave in the number of measurements. This is easy to detect, because all we have to do is to compute the knowledge gradient assuming that we measure alternative x a total of n_x times. The value of information can be nonconcave when the precision of a measurement β^W is low, which means that a single measurement does not contain enough information to accurately indicate whether an alternative is good. Frazier and Powell (2010) identifies this problem and proposes a solution, called the KG(*) policy, where we find n_x that maximizes the *average* value of information over n_x consecutive measurements. We then choose the alternative x with the highest average value, but we may only measure it once.

The knowledge gradient for independent beliefs is appealing because it works well, is easy to compute and there are no tunable parameters such as the parameter z_α for interval estimation. There is, of course, the need to specify a prior, although it is always possible to create a prior using an initial sample (a strategy known as "empirical Bayes").

18.5.2 The knowledge gradient for correlated beliefs

The real power of the knowledge gradient arises in its ability to handle correlations between beliefs. This means that if we observe the value of x, we learn something about x'. Correlated beliefs might arise when $F(x)$ is continuous in x, so that the values of x and x' are similar when x and x' are closer together.

Let Σ^n be the matrix where element (x, x') gives $Cov^n(\mu_x, \mu_{x'})$. Let

$$B^n = (\Sigma^n)^{-1},$$

where B^n is the matrix version of our precision. It is straightforward to show that

$$\begin{aligned} \mu^{n+1} &= (B^{n+1})^{-1}\left(B^n\mu^n + \beta^W W^{n+1} e_{x^n}\right), & (18.23)\\ B^{n+1} &= (B^n + \beta^W e_{x^n}(e_{x^n})^T), & (18.24) \end{aligned}$$

where e_x is a vector of 0's with a 1 in position x. We next compute the matrix form of the conditional change in the variance if we measure alternative x using

$$\begin{aligned} \tilde{\Sigma}^n(x) &= \Sigma^n - \Sigma^{n+1} & (18.25)\\ &= \frac{\Sigma^n e_x (e_x)^T \Sigma^n}{\Sigma^n_{xx} + \lambda^W}, & (18.26) \end{aligned}$$

where $\lambda^W = 1/\beta^W$ is the variance of a measurement. This matrix is analogous to the variance reduction in (18.19). Let

$$\tilde{\sigma}^n(x) = \frac{\Sigma^n e_x}{\sqrt{\Sigma^n_{xx} + \lambda^W}}, \qquad (18.27)$$

be the column vector reflecting the change in the variances across all alternatives if we choose to measure x. The knowledge gradient in the presence of correlated beliefs is given by

$$\begin{aligned} X^{KG}(s) &= \arg\max_x \mathbb{E}\left[\max_i \mu_i^{n+1} \mid S^n = s, x^n = x\right] & (18.28)\\ &= \arg\max_x \mathbb{E}\left[\max_i \left(\mu_i^n + \tilde{\sigma}_i(x^n) Z^{n+1}\right) \mid S^n, x^n = x\right], \end{aligned}$$

where Z^{n+1} is a scalar, standard normal variable with mean 0, variance 1.

There is a way to compute the expectation exactly. We start by defining

$$h(\mu^n, \tilde{\sigma}(x)) = \mathbb{E}\left[\max_i \left(\mu_i^n + \tilde{\sigma}_i(x^n) Z^{n+1}\right) \mid S^n, x^n = x\right]. \qquad (18.29)$$

Substituting (18.29) into (18.28) gives us

$$X^{KG}(s) = \arg\max_x h(\mu^n, \tilde{\sigma}(x)). \qquad (18.30)$$

Let $h(a, b) = \mathbb{E}\max_i(a_i + b_i Z)$, where $a = \mu_i^n$, $b = \tilde{\sigma}_i(\Sigma^n, x^n)$ and Z is our standard normal deviate. The lines $a_i + b_i Z$ can be used to determine the values of Z for which alternative i would become the best, but we have to recognize that some alternatives are simply dominated by all the other alternatives. We first assume that the lines are sorted in order of increasing b_i. It is then necessary to identify these dominated alternatives and eliminate them. This procedure is described in Frazier et al. (2009) and Powell and Ryzhov (2012). Once this step is complete, we can compute the knowledge gradient by using

$$h(a,b) = \sum_{i=1}^{M} (b_{i+1} - b_i) f(-|c_i|),$$

Correlated beliefs arise in a variety of settings. For the purpose of tuning parameters of a function, continuity is the most common motivation because an observation of x changes our belief about nearby points. For a problem where the measurement vector x (or θ) is multidimensional, we can randomly generate a large number of potential points and search over these. We can handle problems with several thousand alternatives, even if we have a measurement budget that is much smaller.

18.5.3 The knowledge gradient for online learning

Up to now, we have focused on developing the knowledge gradient for offline learning problems. It turns out that if we can compute the knowledge gradient for offline learning problems, there is a simple way to adapt this to give us the knowledge gradient for online learning problems. We are not going to duplicate the derivation (given in Ryzhov et al., 2011). If we have a measurement budget of N measurements and we have already completed n, the knowledge gradient for an undiscounted problem is given by

$$\nu_x^{online,n} = \mu_x^n + (N-n)\nu^{KG,n}(x),$$

where $\nu^{KG,n}(x)$ is the offline knowledge gradient for alternative x. If we have a discount factor γ, the knowledge gradient is given by

$$X^{KG,n} = \arg\max_x \mu_x^n + \gamma \frac{1-\gamma^{N-n}}{1-\gamma} \nu_x^{KG,n}.$$

Taking $N \to \infty$, we obtain the knowledge gradient rule for infinite-horizon problems,

$$X^{KG,n} = \arg\max_x \mu_x^n + \frac{\gamma}{1-\gamma} \nu_x^{KG,n}.$$

18.5.4 The knowledge gradient for a parametric belief model

There are many applications where it is simply not practical to use a lookup representation of a belief model. Often, it is useful to use a parametric model. Assume that we would like to approximate $F(x)$ using

$$F(x) \approx \sum_{f \in \mathcal{F}} \eta_f \phi_f(x).$$

Here, instead of one parameter $\mu_x = F(x)$ for each alternative x, we have a vector of parameters $(\eta_f)_{f \in \mathcal{F}}$, where the number of parameters is dramatically smaller than the number of alternatives x (which may in fact be continuous).

It is possible to compute the knowledge gradient when the vector η is the parameters of a linear model. The idea is the same as the knowledge gradient for correlated beliefs using a lookup table. The difference is that rather than computing and storing the matrix Σ^n, which may be computationally prohibitive (imagine a matrix with tens of thousands of rows and columns), we can compute $\tilde{\sigma}^n(x)$ in a much more compact way.

First, create a matrix X^n where each row is made up of $(\phi_1(x^n), \ldots, \phi_f(x^n), \ldots, \phi_F(x^n))$, and where there is a row for each measurement $x^1, \ldots, x^n$. Let Y^n be the column vector of the corresponding observations of the function $F(x)$. We can compute the parameter vector θ using the normal equations, given by

$$\theta^n = [(X^n)^T X^n]^{-1} X^n Y^n.$$

Let $\Sigma^{\eta,n}$ be the covariance matrix for our beliefs about the true values of the vector θ after n observations, which is given by

$$\Sigma^{\eta,n} = [(X^n)^T X^n]^{-1} \sigma_\epsilon^2,$$

where σ_ϵ^2 is the variance of a measurement. It is easy to show that

$$\Sigma^n = X^n \Sigma^{\eta,n} (X^n)^T.$$

However, we do not need to compute the entire matrix Σ^n - we only need a single row corresponding to a particular alternative that we are measuring. So, while this row will still have many elements, at least we do not have to do an entire matrix.

18.5.5 Discussion

We now have a fairly powerful set of tools for performing policy search. We can treat each possible θ in (18.7) as an alternative in an optimal learning problem, and apply the knowledge gradient or other learning policies. We can obtain a realization $W(\omega)$ of the policy value $F^\pi(\theta)$ by simulating the exogenous information driving the dynamic program over a time horizon T, while using the particular vector θ to lock in all decisions via (18.6).

We emphasize that we have not made any attempt to survey the substantial literature that already exists for this problem class, but the knowledge gradient offers some uniquely attractive features: a fast rate of convergence (since it maximizes the value of each measurement) and the ability to handle correlated beliefs, even among large numbers of alternatives. This said, there is a cost to this power. While the knowledge gradient for independent beliefs is quite easy to compute, the knowledge gradient for correlated beliefs requires executing an algorithm. This introduces a level of overhead that will make sense for some applications, but not for others.

18.6 LEARNING WITH A PHYSICAL STATE

The challenge of optimal learning has been well-known in the approximate dynamic programming community for decades, where it is widely referred to as the "exploration vs. exploitation" problem. The problem is easily illustrated using the two-stage

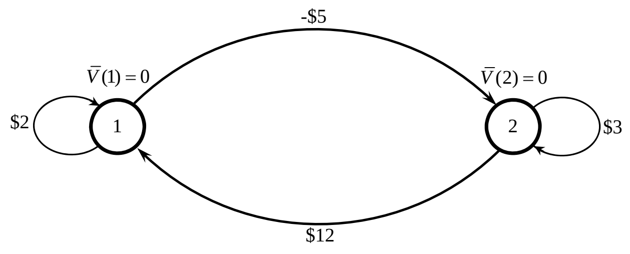

Figure 18.1 A two state dynamic program, where the estimated value of being in each state is currently zero.

dynamic program in Figure 18.1, where we start by approximating the value of each state as zero. If we start in state 1, it appears that it is best to stay in state 1 because transitioning to state 2 would produce an immediate loss of $5, with an approximate future value of zero when we land in state 2. However, only by visiting state 2 do we realize that the approximation is not very good, and we would earn $12 when we make the transition from state 2 to state 1.

Although we previously formulated our learning problem as a dynamic program in Section 18.5, our success with a myopic policy depended on the property that the state variable consisted purely of a belief state. Now, we are considering problems where S_t represents a physical state. In this setting, an action a_t may bring information that allows us to learn, but it also has the effect of moving us from S_t to S_{t+1}, where we now face a different set of actions, and where the information we learned by taking action a_t from state S_t may no longer be useful (we may never return to S_t). The presence of a physical state is so common that references to "dynamic programming" typically imply the existence of a physical state.

We have to be specific about what we are learning in our new dynamic programming setting. These might include

- Costs or rewards - This is what we were doing in our previous discussion of learning about policies. After each action, we may observe a cost which allows us to update our belief about the costs.

- Transition functions - Given a state s and action a, we may transition to a state s' with probability $p(s'|s,a)$. However, we may not be certain about this transition matrix, and as we observe the actual transition, we may update our belief about the transition matrix.

- Value functions - Learning optimal policies may involve learning the value of being in a state. With each transition, we may update our belief about the value of being in a state.

We focus our discussion here on the last category, which is perhaps the one most widely associated with the exploration/exploitation problem.

18.6.1 Heuristic policies

A number of heuristic policies have been suggested to assist with the exploration/exploitation problem in approximate dynamic programming. For example, all of the heuristic policies presented in Section 18.4.3 for problems without a physical state can and have been applied to problems with a physical state.

Several policies have been designed specifically for learning problems with a physical state. One is known as the R-max policy (developed by Brafman and Tennenholtz, 2003), which starts with an upper bound on the value of visiting any state which has been visited fewer than a specified number of times. Needless to say, this policy is limited to problems with relatively small numbers of discrete states and actions.

The E^3 policy ("Explicit Explore or Exploit") is a variation of epsilon-greedy (Kearns and Singh, 2002). Upon reaching state S^n, our decision is made as follows. If S^n is a state that we have never visited before, we make a random decision. If we have visited S^n before, but fewer than m times, we make the decision that we have tried the fewest number of times among all our previous visits to the state. Finally, if we have visited S^n more than m times, we follow an exploitation policy. Once again, the policy reduces to pure exploitation once we have sufficiently explored the state space.

A more sophisticated policy is the local bandit approximation, an adaptation of the Gittins index policy proposed by Duff and Barto (1996) and developed more fully in Ryzhov et al. (2010). If we are in a state s and considering an action a that takes us to a state s', we evaluate this action by estimating the reward earned between starting in s' and then finally terminating back in s, divided by the time required (this is an equivalent formulation of the Gittins index).

18.6.2 The knowledge gradient with a physical state

The knowledge gradient has been adapted to problems with a physical state in Ryzhov and Powell (2010) and Ryzhov and Powell (2011), where the focus is on learning a parametric model of the value function. The idea is as follows. Imagine that we are in state s and considering an action a that might take us to state s' (a random variable given s and a). Once in s' (remember that we have not yet decided if we want to take action a), we identify the best action a' which takes us to another downstream state s''. In the process, we obtain a sample estimate $\hat{v}'$ of the value of being in state s'. If we had obtained this observation, we could use it to update a linear model of the value function, as in (18.5).

We do not have the space to fully derive the knowledge gradient policy with a physical state, but it is useful to take a look at the final result, which is given by

$$\begin{aligned} X^{KG,n}(S^n) &= \max_x \big(C(S^n,x) + \gamma \bar{V}^n(S^{x,n}) \\ &\quad + \sum_{S^{n+1}} p(S^{n+1}|S^n,x)\nu^{KG,n}(S^{x,n},S^{n+1}) \big). \qquad (18.31) \end{aligned}$$

Table 18.1 Performance values in a benchmark energy storage problem (from Ryzhov and Powell, 2011).

	Offline objective		Online objective	
	Mean	Avg. SE	Mean	Avg. SE
Offline KG (basis functions)	1136.20	3.54	-342.23	19.96
Online KG (basis functions)	871.13	3.15	44.58	27.71
Offline KG (lookup)	210.43	0.33	-277.38	15.90
Online KG (lookup)	79.36	0.23	160.28	5.43
Epsilon-greedy (param.)	-475.54	2.30	-329.03	25.31

The policy consists of three terms. The first two include the one-period contribution plus the value of the downstream state, just as we would have with any exploitation policy for dynamic programming. The third term $\nu^{KG,n}(S^{x,n}, S^{n+1})$ captures the value of information arising from being in post-decision state $S^{x,n}$ and then making a stochastic transition to S^{n+1} (in this setting, both $S^{x,n}$ and S^{n+1} represent physical states). This is multiplied by the probability of the transition from $S^{x,n}$ to S^{n+1}, which can be approximated using Monte Carlo simulation if necessary.

Equation (18.31) seems like a natural extension of a basic exploitation policy for approximate dynamic programming to include a value of information term. It is most suitable for online learning problems, which combine the reward being earned (contribution plus value of the next physical state) with the value of the information gained from the subsequent transition. If we compare the knowledge gradient for offline and online settings, we see that the knowledge gradient policy for offline learning would be given by

$$X^{Off,n}(S^n) = \arg\max_x \sum_{S^{n+1}} p\left(S^{n+1}|S^n, x\right) \nu^{KG,n}\left(S^{x,n}, S^{n+1}\right). \qquad (18.32)$$

Table 18.1 presents comparisons between a series of algorithms in both an offline setting (where we use training iterations to learn a value function approximation, and then evaluate the policy using a separate set of testing iterations) and an online setting (where we accumulate rewards as we go). We compare both online and offline KG, using both a lookup table representation as well as basis functions, against an epsilon-greedy policy. We note the online version of KG works best when evaluated using an online objective function (as we would expect), while the offline version of KG works best when we use an offline objective function. Both outperform by a significant margin the epsilon-greedy policy when using a parametric value function approximation.

Bibliography

Auer, P., Cesa-bianchi, N. and Fischer, P. (2002), ‘Finite time analysis of the multi-armed bandit problem’, *Machine Learning* **47**(2-3), 235–256.

Bertsekas, D. P. (2011*a*), Approximate Dynamic Programming, *in* ‘Dynamic Programming and Optimal Control 3rd Edition, Volume II’, 3 edn, Vol. II, Athena Scientific, Belmont, MA, chapter 6.

Bertsekas, D. P. (2011*b*), ‘Approximate policy iteration: a survey and some new methods’, *Journal of Control Theory and Applications* **9**(3), 310–335.

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

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

Brafman, R. I. and Tennenholtz, M. (2003), ‘R-max – a general polynomial time algorithm for near-optimal reinforcement learning’, *Journal of Machine Learning Research* **3**, 213–231.

Bubeck, S., Munos, R. and Stoltz, G. (2009), Pure Exploration in Multi-Armed Bandits Problems, *in* ‘Proceedings of the 20th International Conference on Algorithmic Learning Theory’.

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

Chick, S. E. and Gans, N. (2009), 'Economic analysis of simulation selection problems', *Management Science* **55**(3), 421–437.

Coulom, R. (2006), Efficient selectivity and backup operators in monte-carlo tree search, *in* 'Proceedings of the 5th International Conference on Computers and Games', pp. 72–83.

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

Duff, M. and Barto, A. (1996), Local bandit approximation for optimal learning problems, *in* M. Mozer, M. Jordan and T. Pesche, eds, 'Advances in Neural Information Processing Systems', Vol. 9, Cambridge, MA: MIT Press, pp. 1019–1025.

Frazier, P. I. and Powell, W. B. (2010), 'Paradoxes in Learning and the Marginal Value of Information', *Decision Analysis* **7**(4), 378–403.

Frazier, P. I., Powell, W. B. and Dayanik, S. (2009), 'The Knowledge-Gradient Policy for Correlated Normal Beliefs', *INFORMS Journal on Computing* **21**(4), 599–613.

Fu, M. C. (2008), 'What You Should Know About Simulation and Derivatives', *Naval Research Logistics* **55**(8), 723–736.

Gelly, S. and Wang, Y. (2006), Exploration exploitation in go: UCT for Monte-Carlo go, *in* 'Twentieth Annual Conference on Neural Information Processing Systems (NIPS 2006)', Citeseer.

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

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

Kaelbling, L. P. (1993), *Learning in embedded systems*, MIT Press, Cambridge, MA.

Kearns, M. and Singh, S. (2002), 'Near-optimal reinforcement learning in polynomial time', *Machine Learning* **49**(2), 209–232.

Lai, T. L. and Robbins, H. (1985), 'Asymptotically Efficient Adaptive Allocation Rules', *Advances in Applied Mathematics* **6**, 4–22.

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

Powell, W. B. and Ryzhov, I. O. (2012), *Optimal Learning*, John Wiley & Sons Inc.

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

Robbins, H. and Monro, S. (1951), 'A stochastic approximation method', *The Annals of Mathematical Statistics* **22**(3), 400–407.

Ruszczynski, A. and Shapiro, A. (2003), *Handbooks in Operations Research and Management Science: Stochastic Programming*, Vol. 10, Elsevier, Amsterdam.

Ryzhov, I. O. and Powell, W. B. (2010), Approximate Dynamic Programming With Correlated Bayesian Beliefs, *in* 'Proceedings of the 48th Allerton Conference on Communication, Control, and Computing', pp. 1360–1367.

Ryzhov, I. O. and Powell, W. B. (2011), Bayesian Active Learning With Basis Functions, *in* 'Proceedings of the 2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning', pp. 143–150.

Ryzhov, I. O., Powell, W. B. and Frazier, P. I. (2011), 'The knowledge gradient algorithm for a general class of online learning problems', *Operations Research (to appear)* .

Ryzhov, I. O., Valdez-Vivas, M. R. and Powell, W. B. (2010), Optimal Learning of Transition Probabilities in the Two-Agent Newsvendor Problem, *in* B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan and E. Yücesan, eds, 'Proceedings of the 2010 Winter Simulation Conference', pp. 1088–1098.

Shapiro, A., Dentcheva, D. and Ruszczynski, A. (2009), *Lectures on stochastic programming: modeling and theory*, SIAM, Philadelphia.

Silver, D. (2009), Reinforcement Learning and Simulation-Based search in Computer Go, PhD thesis, University of Alberta.

Singh, S. P., Jaakkola, T., Szepesvari, C. and Littman, M. (2000), 'Convergence results for single-step on-policy reinforcement-learning algorithms', *Machine Learning* **38**(3), 287—-308.

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

Sutton, R., Maei, H., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, C. and Wiewiora, E. (2009), Fast gradient-descent methods for temporal-difference learning with linear function approximation, *in* 'Proceedings of the 26th International Conference on Machine Learning', pp. 993–1000.

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

Szepesvari, C. (2010), 'Algorithms for Reinforcement Learning', *Synthesis Lectures on Artificial Intelligence and Machine Learning* **4**(1), 1–103.