Reinforcement Learning and Optimal Control

by

Dimitri P. Bertsekas

2019

Chapter 2
Approximation in Value Space

SELECTED SECTIONS

WWW site for book information and orders

http://www.athenasc.com

Athena Scientific, Belmont, Massachusetts

2

Approximation in Value Space

Contents

As we noted in Chapter 1, the exact solution of optimal control problems by DP is often impossible. To a great extent, the reason lies in what Bellman has called the "curse of dimensionality." This refers to a rapid increase of the required computation and memory storage as the size of the problem increases. Moreover, there are many circumstances where the structure of the given problem is known well in advance, but some of the problem data, such as various system parameters, may be unknown until shortly before control is needed, thus seriously constraining the amount of time available for the DP computation. These difficulties motivate suboptimal control schemes that strike a reasonable balance between convenient implementation and adequate performance.

The philosophy that guides our subsequent presentation is based on the author's view of the present state of the art in the field: there are no RL methods that are guaranteed to work for all or even most DP problems, but there are enough methods to try on a given problem with a reasonable chance of success in the end. With this in mind, we will present a broad variety of methods, aiming to provide intuition into their inner workings, as well as an understanding of their analytical and computational properties. We present some analysis, usually based on DP principles, but we do not provide solid performance guarantees for most of our methods. We also discuss the potential merits of different methods within particular practical contexts, but our discussion is speculative and should be taken with a grain of salt and/or adapted to the context at hand.

In this chapter, we lay out a broad landscape of methods for approximation in the contexts of the finite horizon deterministic and stochastic DP problems of Chapter 1, and then focus on approximation in value space. Many of the finite horizon methods will be adapted to infinite horizon DP later, together with some additional methods that are specific to the infinite horizon context.

2.1 APPROXIMATION APPROACHES IN REINFORCEMENT LEARNING

There are two general types of approximation in DP-based suboptimal control. The first is *approximation in value space*, where we aim to approximate the optimal cost function. The second is *approximation in policy space*, where we select the policy by using optimization over a suitable class of policies. In this section we provide a broad overview of these approximation approaches.

Approximation in Value Space

In approximation in value space, we approximate the optimal cost-to-go functions J_k^* with some other functions $\tilde{J}_k$. We then replace J_k^* in the DP

equation with $\tilde{J}_k$. In particular, at state x_k, we use the control obtained from the minimization

$$\tilde{\mu}_k(x_k) \in \arg \min_{u_k \in U_k(x_k)} E\Big\{ g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}\big(f_k(x_k, u_k, w_k)\big) \Big\}. \quad (2.1)$$

This defines a suboptimal policy $\{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\}$. There are several possibilities for selecting or computing the functions $\tilde{J}_k$, which are discussed in this chapter, and also in subsequent chapters.

Note that the expected value expression appearing in the right-hand side of Eq. (2.1) can be viewed as an approximate Q-factor,

$$\tilde{Q}_k(x_k, u_k) = E\Big\{ g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}\big(f_k(x_k, u_k, w_k)\big) \Big\},$$

and the minimization in Eq. (2.1) can be written as

$$\tilde{\mu}_k(x_k) \in \arg \min_{u_k \in U_k(x_k)} \tilde{Q}_k(x_k, u_k),$$

(cf. Section 1.2). This also suggests a variant of approximation in value space, which is based on using Q-factor approximations that may be obtained directly, i.e., without the intermediate step of obtaining the cost function approximations $\tilde{J}_k$. In what follows in this chapter, we will focus on cost function approximation, but we will occasionally digress to discuss direct Q-factor approximation.

Approximation in Value Space - Multistep Lookahead

Approximation in value space based on the minimization (2.1) is commonly referred to as *one-step lookahead*, because the future costs are approximated by $\tilde{J}_{k+1}$, after a single step. An important variation is *multistep lookahead*, whereby at state x_k we minimize the cost of the first $\ell > 1$ stages with the future costs approximated by a function $\tilde{J}_{k+\ell}$. For example, in two-step lookahead the function $\tilde{J}_{k+1}$ is given by

$$\tilde{J}_{k+1}(x_{k+1}) = \min_{u_{k+1} \in U_{k+1}(x_{k+1})} E\Big\{ g_{k+1}(x_{k+1}, u_{k+1}, w_{k+1}) + \tilde{J}_{k+2}\big(f_{k+1}(x_{k+1}, u_{k+1}, w_{k+1})\big) \Big\},$$

where $\tilde{J}_{k+2}$ is some approximation of the optimal cost-to-go function J^*_{k+2}.

Actually, as the preceding two-step lookahead case illustrates, one may view ℓ-step lookahead as the special case of one-step lookahead where the lookahead function is the optimal cost function of an $(\ell - 1)$-stage DP problem with a terminal cost $(\tilde{J}_{k+\ell})$ at the end of the $\ell - 1$ stages. However, it is often important to discuss ℓ-step lookahead separately, in

order to address special implementation issues that do not arise in the context of one-step lookahead.

The motivation for ℓ-step lookahead is that for increasing values of ℓ, one may require a less accurate approximation $\tilde{J}_{k+\ell}$ to obtain good performance. Otherwise expressed, for the same quality of cost function approximation, better performance may be obtained as ℓ becomes larger. This makes intuitive sense, since in this case, the cost of more stages is treated exactly. Indeed this expectation is typically realized in practice, although one can construct artificial examples when this is not so (see Section 2.2.1). Generally, one should at least try to use the largest value of ℓ for which the overhead for solving the ℓ-step lookahead minimization problem on-line is acceptable.

In our initial discussion of approximation in value space of Section 2.1, we will focus on one-step lookahead. There are straightforward extensions of the main ideas to the multistep context, which we will discuss in Section 2.2.

Approximation in Policy Space

The major alternative to approximation in value space is *approximation in policy space*, whereby we select the policy from a suitably restricted class of policies, usually a parametric class of some form. In particular, we can introduce a parametric family of policies

$$\mu_k(x_k, r_k), \qquad k = 0, \ldots, N-1,$$

where r_k is a parameter, such as a family represented by a neural network, and then estimate the parameters r_k using some type of optimization.

An important advantage of approximation in policy space is that the computation of controls during on-line operation of the system is often much easier compared with the lookahead minimization (2.1). However, this advantage can also be gained by combining approximation in value space with policy approximation, as we will describe in Section 2.1.5.

In this chapter we discuss primarily approximation in value space, although some of the ideas are also relevant to approximation in policy space. We focus on finite horizon problems, postponing the discussion of infinite horizon problems for Chapter 4 and later. However, the finite horizon ideas are relevant to the infinite horizon setting, and many of the methods of the present chapter and Chapter 3 also apply with small modifications to infinite horizon problems.

Model-Based Versus Model-Free Implementation

Generally, a finite horizon DP problem is defined by the state, control, and disturbance spaces, the functions f_k and g_k, the control constraint sets

$U_k(x_k)$, and the probability distributions of the disturbances. We refer to these as the *mathematical model* of the problem. It is important to note that there is only one mathematical model for a given problem, but there may be several different implementations of a given method for solving the problem exactly or approximately. Some of these implementations may rely exclusively on analytical calculations using the mathematical model, but others may additionally or exclusively rely on Monte Carlo simulation.

In this book, we will use an unambiguous technical definition of a model-free method (or more accurately a model-free implementation of a given method).† In particular, for us the key relevant attribute of an implementation is whether an analytical formula-based calculation or a Monte Carlo simulation is used to compute expected values such as those arising in one-step and multistep lookahead expressions. We thus distinguish between two types of implementations of the various methods given in this book:

(a) In the *model-based* case, we assume that the conditional probability distribution of w_k, given (x_k, u_k), is available in closed form. By this we mean that the value of $p_k(w_k \mid x_k, u_k)$ is available for any triplet (x_k, u_k, w_k). Moreover, the functions g_k and f_k are also available. In a model-based implementation, expected values, such as the one in the lookahead expression of Eq. (2.1), are obtained with algebraic calculations as opposed to Monte Carlo simulation.

(b) In the *model-free* case, the calculation of the expected value in the expression of Eq. (2.1), and other related expressions, is done with Monte Carlo simulation. There may be two possible reasons for this.

  (1) An analytical expression of the probabilities $p_k(w_k \mid x_k, u_k)$ is not available, but instead there is a computer program/simulator that for any given state x_k and control u_k, simulates sample probabilistic transitions to a successor state x_{k+1}, and generates the corresponding transition costs. In this case the expected value can be computed approximately by Monte Carlo simulation.‡

  (2) The probabilities $p_k(w_k \mid x_k, u_k)$ are available for any triplet (x_k, u_k, w_k), but for reasons of computational efficiency we pre-

† There is considerable variance in the use of the term "model-free" in the literature. For example, some authors define model-based RL methods as those that estimate costs-to-go of states, and model-free those that just estimate Q-factors of state-control pairs.

‡ The term Monte Carlo simulation mostly refers to the use of a software simulator. However, a hardware or a combined hardware/software simulator may also be used in some practical situations to generate samples that are used for Monte Carlo averaging.

fer to compute the expected value in the expression (2.1) by using sampling and Monte Carlo simulation. Thus the expected value is computed as in the case where an analytical expression of the probabilities $p_k(w_k \mid x_k, u_k)$ is not available, but instead there is a computer simulator.†

Note that for deterministic problems there is no expected value to compute, so methods for these problems typically come under the model-based category, even if values of the functions g_k and f_k become available through complicated computer calculations. Still however, Monte Carlo simulation may enter the solution process of a deterministic problem for a variety of reasons. For example the games of chess and Go are perfectly deterministic, but the AlphaGo and AlphaZero programs (Silver et al. [SHM16], [SHS17]) use randomized policies and rely heavily on Monte Carlo tree search techniques, which use sampling and will be discussed in Section 2.4.2. The same is true for some policy gradient methods, which will be discussed later.

To summarize, *the use of sampling and Monte Carlo simulation is the defining attribute for an implementation to be model-based or model-free in the terminology of this book*. This view of a model-free approach aims to avoid ambiguities in cases where the mathematical model is available to use for closed form calculations, but Monte Carlo simulation is used anyway for reasons of convenience or computational efficiency.

2.1.1 General Issues of Approximation in Value Space

There are two major issues in a value space approximation scheme, and each of the two can be considered separately from the other:

(1) *Obtaining* $\tilde{J}_k$, i.e., the method to compute the lookahead functions $\tilde{J}_k$ that are involved in the lookahead minimization (2.1). There are quite a few approaches here (see Fig. 2.1.1). Several of them are discussed in this chapter, and more will be discussed in subsequent chapters.

(2) *Control selection*, i.e., the method to perform the minimization (2.1) and implement the suboptimal policy $\tilde{\mu}_k$. Again there are several exact and approximate methods for control selection, some of which will be discussed in this chapter (see Fig. 2.1.1).

In this section we will provide a high level discussion of these issues, focusing exclusively on the case of one-step lookahead.

† The idea of using Monte Carlo simulation to compute complicated integrals or even sums of many numbers is used widely in various types of numerical computations. It encompasses efficient Monte Carlo techniques known as *Monte Carlo integration* and *importance sampling*; see textbooks such as [Liu01], [AsG10], [RoC10], [Gla13] for detailed developments.

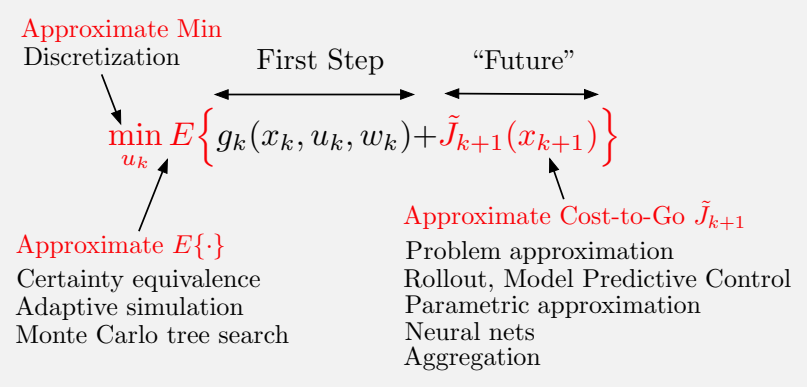

Figure 2.1.1 Schematic illustration of various options for approximation in value space with one-step lookahead. The lookahead function values $\tilde{J}_{k+1}(x_{k+1})$ approximate the optimal cost-to-go values $J^*_{k+1}(x_{k+1})$, and can be computed by a variety of methods. There may be additional approximations in the minimization over u_k and in the computation of the expected value over w_k; see Section 2.1.1.

Regarding the computation of $\tilde{J}_k$, we will consider four types of methods:

(a) *Problem approximation (Section 2.3)*: Here the functions $\tilde{J}_k$ in Eq. (2.1) are obtained as the optimal or nearly optimal cost functions of a simplified optimization problem, which is more convenient for computation. Simplifications may include, exploiting decomposable structure, ignoring various types of uncertainties, and reducing the size of the state space. Another form of simplification is *aggregation*, which is discussed separately in Chapter 6.

(b) *On-line approximate optimization (Sections 2.4, 2.5)*: These methods often involve the use of a suboptimal policy or heuristic, which is applied on-line when needed to approximate the true optimal cost-to-go values. The suboptimal policy may be obtained by any other method, e.g., problem approximation. *Rollout algorithms* and *model predictive control* are prime examples of these methods.

(c) *Parametric cost approximation (Chapter 3)*: Here the functions $\tilde{J}_k$ in Eq. (2.1) are obtained from a given parametric class of functions $\tilde{J}_k(x_k, r_k)$, where r_k is a parameter vector, selected by a suitable algorithm. The parametric class is typically obtained by using prominent characteristics of x_k called *features*, which can be obtained either through insight into the problem at hand, or by using training data and some form of neural network.

(d) *Aggregation (Chapter 6)*: This is a special but rather sophisticated form of problem approximation. A simple example is to select a set

of representative states for each stage, restrict the DP algorithm to these states only, and approximate the costs-to-go of other states by interpolation between the optimal costs-to-go of the representative states. In another example of aggregation, the state space is divided into subsets, and each subset is viewed as a state of an "aggregate DP problem." The functions $\tilde{J}_k$ are then derived from the optimal cost functions of the aggregate problem. The state space partition can be arbitrary, but is often determined by using features (states with "similar" features are grouped together). Moreover, aggregation can be combined in complementary fashion with the methods (a)-(c) above, and can use as a starting point an approximate cost-to-go function produced by any one of these methods; e.g., apply a parametric approximation method and enhance the resulting cost function through local corrections obtained by aggregation.

Additional variations of the above methods are obtained when used in combination with approximate minimization over u_k in Eq. (2.1), and also when the expected value over w_k is computed approximately via a certainty equivalence approximation (cf. Section 2.3.2) or adaptive simulation and Monte Carlo tree search (cf. Section 2.4.2).

2.1.2 Off-Line and On-Line Methods

In approximation in value space an important consideration is whether the cost-to-go functions $\tilde{J}_{k+1}$ and the corresponding suboptimal policy $\{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\}$ of Eq. (2.1) are computed *off-line* (i.e., before the control process begins, and for all x_k and k), or *on-line* (i.e., after the control process begins, when needed, and for just the states x_k to be encountered).

Usually, for challenging problems, the controls $\tilde{\mu}_k(x_k)$ are computed on-line, since their storage may be difficult when the state space is large. However, the on-line or off-line computation of $\tilde{J}_{k+1}$ is an important design choice. We thus distinguish between:

(i) *Off-line methods*, where the entire function $\tilde{J}_{k+1}$ in Eq. (2.1) is computed for every k, before the control process begins. The values $\tilde{J}_{k+1}(x_{k+1})$ are either stored in memory or can be obtained with a simple and fast computation, as needed in order to compute controls by one-step lookahead. The advantage of this is that most of the computation is done off-line, before the first control is applied at time 0. Once the control process starts, no extra computation is needed to obtain $\tilde{J}_{k+1}(x_{k+1})$ for implementing the corresponding suboptimal policy.

(ii) *On-line methods*, where most of the computation is performed just after the current state x_k becomes known, the values $\tilde{J}_{k+1}(x_{k+1})$ are computed only at the relevant next states x_{k+1}, and are used to compute the control to be applied via Eq. (2.1). In contrast with the

off-line approximation methods, these methods are well-suited for *on-line replanning*, whereby the problem data may change over time. A similar situation where on-line methods may have an advantage is when the initial state and other problem data may become known just before the control process begins.

Examples of typically off-line schemes are neural network and other parametric approximations, as well as aggregation. Examples of typically on-line schemes are rollout and model predictive control. Schemes based on problem approximation may be either on-line or off-line depending on other problem-related factors. Of course there are also hybrid methods, where significant computation is done off-line to expedite the on-line computation of needed values of $\tilde{J}_{k+1}$. An example is the truncated rollout schemes to be discussed in Sections 2.4 and 5.1.

2.1.3 Model-Based Simplification of the Lookahead Minimization

We will now consider ways to facilitate the calculation of the suboptimal control $\tilde{\mu}_k(x_k)$ at state x_k via the minimization of the one-step lookahead expression

$$E\Big\{g_k(x_k,u_k,w_k)+\tilde{J}_{k+1}\big(f_k(x_k,u_k,w_k)\big)\Big\}, \tag{2.2}$$

once the cost-to-go approximating functions $\tilde{J}_{k+1}$ have been selected. In this section, we will assume that we have a mathematical model, i.e., that the functions g_k and f_k are available in essentially closed form, and that the conditional probability distribution of w_k, given (x_k,u_k), is also available. Moreover, Monte Carlo simulation is not used to compute the expected value in Eq. (2.2). We will address the model-free case in the next section.

Important issues here are the computation of the expected value (if the problem is stochastic) and the minimization over $u_k \in U_k(x_k)$ in Eq. (2.2). Both of these operations may involve substantial work, which is of particular concern when the minimization is to be performed on-line.

One possibility to eliminate the expected value from the expression (2.2) is (assumed) *certainty equivalence*. Here we choose a typical value $\tilde{w}_k$ of w_k, and use the control $\tilde{\mu}_k(x_k)$ that solves the deterministic problem

$$\min_{u_k\in U_k(x_k)}\Big[g_k(x_k,u_k,\tilde{w}_k)+\tilde{J}_{k+1}\big(f_k(x_k,u_k,\tilde{w}_k)\big)\Big]. \tag{2.3}$$

The approach of turning a stochastic problem into a deterministic one by replacing uncertain quantities with single typical values highlights the possibility that $\tilde{J}_{k+1}$ may itself be obtained by using deterministic methods. We will discuss this approach and its variations in greater detail later in this chapter (see Section 2.3).

Let us now consider the issue of algorithmic minimization over $U_k(x_k)$ in Eqs. (2.2) and (2.3). If $U_k(x_k)$ is a finite set, the minimization can be

done by brute force, through exhaustive computation and comparison of the relevant cost expressions. This of course can be very time consuming, particularly for multistep lookahead, but parallel computation can be used with great effect for this purpose [as well as for the calculation of the expected value in the expression (2.2)]. For some discrete control problems, *integer programming* techniques may also be used. Moreover, for deterministic problems with multistep lookahead, sophisticated exact or approximate *shortest path methods* may be considered; several methods of this type are available, such as label correcting methods, A^* methods, and their variants (see the author's textbooks [Ber98] and [Ber17] for detailed accounts, which are consistent with the context of this chapter).

When the control constraint set is infinite, it may be replaced by a finite set through discretization. However, a more efficient alternative may be to use continuous space *nonlinear programming* techniques. This possibility can be attractive for deterministic problems, which lend themselves better to continuous space optimization; an example is the model predictive control context (see Section 2.5).

For stochastic problems with continuous control spaces and either one-step or multistep lookahead, the methodology of *stochastic programming* may be useful. This methodology bears a close connection with linear and nonlinear programming methods. We refer to the textbook [Ber17] for a discussion of its application to the approximate DP context, and references to the literature. Still another possibility to simplify the one-step lookahead minimization (2.2) is based on Q-factor approximation, which is also suitable for model-free policy implementation, as we discuss next.

2.1.4 Model-Free Q-Factor Approximation in Value Space

One of the major aims of this book is to discuss methods where a mathematical model [the system functions f_k, the probability distribution of w_k, and the one-stage cost functions g_k] is not used because it is either hard to construct, or simply inconvenient. We assume instead that the system and cost structure can be simulated in software far more easily (think, for example, control of a queueing network with complicated but well-defined service disciplines at the queues).†

In this section, we will review some of the high-level ideas of passing from model-based to model-free policy implementations for stochastic problems. In particular, we assume that:

(a) There is a computer program/simulator that for any given state x_k and control $u_k \in U_k(x_k)$, simulates sample probabilistic transitions to a successor state x_{k+1}, and generates the corresponding transition costs.

† Another possibility is to use the real system to provide the next state and transition cost, but we will not deal explicitly with this case in this book.

(b) A cost function approximation $\tilde{J}_{k+1}$ is available. Approaches to obtain $\tilde{J}_{k+1}$ in model-free fashion will be discussed in the context of specific methods later. For example $\tilde{J}_{k+1}$ may be obtained by solving a simpler problem for which a model is available, or it may be separately obtained without a mathematical model, by using a simulator.

We want to use the functions $\tilde{J}_{k+1}$ and the simulator to compute or approximate the Q-factors

$$E\Big\{g_k(x_k,u_k,w_k)+\tilde{J}_{k+1}\big(f_k(x_k,u_k,w_k)\big)\Big\},$$

for all $u_k \in U_k(x_k)$, and then find the minimal Q-factor and corresponding one-step lookahead control.

Given a state x_k, we may use the simulator to compute these Q-factors for all the pairs (x_k,u_k), $u_k \in U_k(x_k)$, and then select the minimizing control. However, in many cases this can be very time-consuming. To deal with this difficulty, we may introduce a parametric family/approximation architecture of Q-factor functions,

$$\tilde{Q}_k(x_k,u_k,r_k),$$

where r_k is the parameter vector and use a least squares fit/regression to approximate the expected value that is minimized in Eq. (2.2). One possibility is to use a neural network parametric architecture; see Chapter 3, where we discuss methods for selecting and training parametric architectures. The steps are as follows:

Summary of Q-Factor Approximation Based on Approximation in Value Space

Assume that the value of $\tilde{J}_{k+1}(x_{k+1})$ is available for any given x_{k+1}:

(a) Use the simulator to collect a large number of "representative" quadruplets $(x_k^s,u_k^s,x_{k+1}^s,g_k^s)$, and corresponding Q-factors

$$\beta_k^s = g_k^s + \tilde{J}_{k+1}(x_{k+1}^s), \qquad s=1,\ldots,q. \tag{2.4}$$

Here x_{k+1}^s is the simulator's output of the next state

$$x_{k+1}^s = f_k(x_k^s,u_k^s,w_k^s)$$

that corresponds to some disturbance w_k^s. This disturbance also determines the one-stage-cost sample

$$g_k^s = g_k(x_k^s,u_k^s,w_k^s).$$

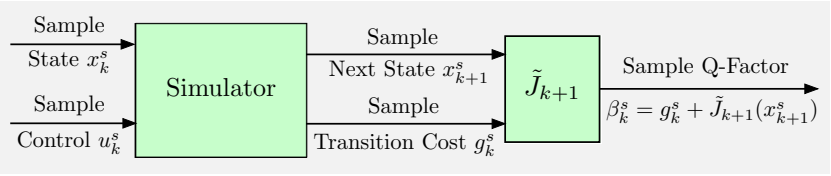

Figure 2.1.2 Schematic illustration of the simulator used for a model-free Q-factor approximation, assuming approximate cost functions $\tilde{J}_{k+1}$ are known. The input to the simulator are sample state-control pairs (x_k^s, u_k^s), and the outputs are a next state sample x_{k+1}^s and cost sample g_k^s. These correspond to a disturbance w_k^s according to

$$x_{k+1}^s = f_k(x_k^s, u_k^s, w_k^s), \qquad g_k^s = g_k(x_k^s, u_k^s, w_k^s).$$

The actual value of w_k^s need not be output by the simulator. The sample Q-factors β_k^s are generated according to Eq. (2.4), and are used in the least squares regression (2.5) to yield a parametric Q-factor approximation $\tilde{Q}_k$ and the policy implementation (2.6).

The simulator need not output w_k^s; only the sample next state x_{k+1}^s and sample cost g_k^s are needed (see Fig. 2.1.2). Moreover, the simulator may output β_k^s directly, assuming it has the means to compute it.

(b) Compute the parameter $\bar{r}_k$ by the least-squares regression

$$\bar{r}_k \in \arg\min_{r_k} \sum_{s=1}^{q} \big(\tilde{Q}_k(x_k^s, u_k^s, r_k) - \beta_k^s\big)^2. \tag{2.5}$$

(c) Use the policy

$$\tilde{\mu}_k(x_k) \in \arg\min_{u_k \in U_k(x_k)} \tilde{Q}_k(x_k, u_k, \bar{r}_k). \tag{2.6}$$

Note some important points about the preceding procedure:

(1) It is model-free in the sense that it is based on Monte Carlo simulation. Moreover, it does not need the functions f_k and g_k, and the probability distribution of w_k to generate the policy $\tilde{\mu}_k$ through the least squares regression (2.5) and the Q-factor minimization (2.6). Using the simulator to collect the samples (2.4) and the cost function approximation $\tilde{J}_{k+1}$ suffices.

(2) Two approximations are potentially required: One to compute $\tilde{J}_{k+1}$,

which is needed for the samples β_k^s [cf. Eq. (2.4)], and another to compute $\tilde{Q}_k$ through the regression (2.5). The approximation methods to obtain $\tilde{J}_{k+1}$ and $\tilde{Q}_k$ may be unrelated.

(3) The policy $\tilde{\mu}_k$ obtained through the minimization (2.6) is not the same as the one obtained through the minimization (2.2). There are two reasons for this. One is the approximation error introduced by the Q-factor architecture $\tilde{Q}_k$, and the other is the simulation error introduced by the finite-sample regression (2.5). We have to accept these sources of error as the price to pay for the convenience of not requiring a mathematical model for policy implementation.

Let us also mention a variant of the least squares minimization in Eq. (2.5), which is to use a *regularized minimization* where a quadratic regularization term is added to the least squares objective. This term is a multiple of the squared deviation $\|r - \hat{r}\|^2$ of r from some initial guess $\hat{r}$. Moreover, in some cases, a nonquadratic minimization may be used in place of Eq. (2.5) to determine $\bar{r}_k$, but in this book we will focus on least squares exclusively.

2.1.5 Approximation in Policy Space on Top of Approximation in Value Space

A common approach for approximation in policy space, is to introduce a parametric family of policies $\tilde{\mu}_k(x_k, r_k)$, where r_k is a parameter vector. The parametrization may involve a neural network as we will discuss in Chapter 3. Alternatively, the parametrization may involve problem-specific features, exploiting the special structure of the problem at hand.

A general scheme for parametric approximation in policy space is to obtain a large number of sample state-control pairs (x_k^s, u_k^s), $s = 1, \ldots, q$, such that for each s, u_k^s is a "good" control at state x_k^s. We can then choose the parameter r_k by solving the least squares/regression problem

$$\min_{r_k} \sum_{s=1}^{q} \left\| u_k^s - \tilde{\mu}_k(x_k^s, r_k) \right\|^2 \tag{2.7}$$

(possibly with added regularization).† In particular, we may determine u_k^s using a human or a software "expert" that can choose "near-optimal"

† It is implicitly assumed here (and in similar situations later) that the controls are members of a Euclidean space so that the distance between two controls can be measured by their normed difference. Otherwise the norm should be replaced by some distance metric within the control space. Regression problems of this type arise in the training of *parametric classifiers* based on data, including the use of neural networks (see Section 3.5). Assuming a finite control space, the classifier is trained using the data $\left(x_k^s, u_k^s\right)$, $s = 1, \ldots, q$, which are viewed as state-category pairs, and then a state x_k is classified as being of "category"

controls at given states, so $\tilde{\mu}_k$ is trained to match the behavior of the expert. Methods of this type are commonly referred to as *supervised learning* in artificial intelligence (see also the discussion in Section 5.7.2).

A special case of the above procedure, which connects with approximation in value space, is to generate the sample state-control pairs (x_k^s, u_k^s) through a one-step lookahead minimization of the form

$$u_k^s \in \arg\min_{u \in U_k(x_k)} E\Big\{g_k(x_k^s, u, w_k) + \tilde{J}_{k+1}\big(f_k(x_k^s, u, w_k)\big)\Big\}, \tag{2.8}$$

where $\tilde{J}_{k+1}$ is a suitable (separately obtained) approximation in value space; cf. Eq. (2.2), or an approximate Q-factor based minimization

$$u_k^s \in \arg\min_{u_k \in U_k(x_k^s)} \tilde{Q}_k(x_k^s, u_k, \bar{r}_k), \tag{2.9}$$

[cf. Eq. (2.6)]. In this case, we collect the sample state-control pairs (x_k^s, u_k^s), $s = 1, \ldots, q$, by using approximation in value space through Eq. (2.8) or Q-factor approximation through Eq. (2.9), and then apply approximation in policy space through Eq. (2.7) (i.e., approximation in policy space is built on top of approximation in value space).

A major advantage of schemes based on the minimization (2.7) is that once the parametrized policy is obtained, the on-line implementation of the policy is fast and does not involve extensive calculations such as minimizations of the form (2.8) or (2.9). This advantage is generally shared by schemes that are based on approximation in policy space.

$\tilde{\mu}_k(x_k, r_k)$. Parametric approximation architectures, and their training through the use of data and regression techniques are described in Chapter 3.

2.6 NOTES AND SOURCES

When considering approximations within a large and challenging problem setting, a primary question is what to approximate. In this chapter, and indeed in the entire book, we argue that *in RL the key objects to approximate are values and policies*, leading to a broad division between approximation in value space and approximation in policy space approaches.

The structure of approximation in value space involves limited lookahead minimization, and is depicted in the key Figs. 2.1.1 and 2.2.1. There we have argued that there are three candidate areas for approximation to consider:

(a) *Cost-to-go function approximation*, which defines the approximate Q-factors at a given state.

(b) *Simplification of expected values* involved in the calculation of the approximate Q-factors.

(c) *Simplification of the Q-factor minimization* over all admissible controls.

There are several candidate approaches for each of these three approximations. Indeed most of our subsequent account revolves around the development of the relevant algorithmic machinery and the insight needed to bring to bear the right combination of approximations on a given problem. Significantly, the three approximations are largely decoupled from each other, allowing a broad spectrum of mixtures of algorithmic choices.

One more choice in approximation in value space is important, namely *whether to use one-step or multistep lookahead minimization* (also whether to try MCTS or not). A simple practical guideline is to use the longest lookahead that the computational budget restrictions will allow. The reason is that, except for unusual circumstances, longer lookahead will typically yield at least some improvement in performance. Still, however, it may be important to use selectively long lookahead; this may be used to properly allocate computational resources during on-line lookahead optimization. Some insight about this issue may be drawn from the game of chess, where the use of long lookahead is very beneficial; generally, it is widely recognized that chess positions that are dynamic and involve principal lines of play call for longer lookahead. This is consistent with MCTS and suggests to extend the lookahead at states that seem promising (are likely to occur under an optimal policy, based on the interim results of computation), and also involve critical decisions (committing decisions whose consequences are "significant" and/or "irreversible"). Thus, we may try selectively to extend the lookahead and/or improve the quality of the cost function approximation at such states, assuming of course that we can recognize them in some way (which will be likely problem-dependent).

Approximation in policy space is based on somewhat different algo-

rithmic ideas than approximation in value space: optimization within a restricted class of parametrized policies. It requires a methodological viewpoint that is less connected with DP, and it will receive less attention in this book. Still, approximation in policy space is an important approach that may be favored in circumstances to be explained in Section 5.7. Moreover it may be combined with approximation in value space, as we have already noted in Section 2.1.5, and we will discuss further in Section 5.7.

In Section 1.5 we provided a list of many exact DP and approximate DP/RL textbooks and survey papers. In what follows in this section, and in other similar end-of-chapter sections, we will aim at a more targeted if inevitably incomplete set of citations to specific research topics.

Sections 2.1, 2.2: Approximation in value space has been considered in an ad hoc manner since the early days of DP, motivated by the curse of dimensionality. The idea was reframed and coupled with model-free simulation methods that originated in the 1980s in artificial intelligence.

Section 2.3: The problem approximation approach has a long history in optimal control and operations research. The author's book [Ber17], Section 6.2.1, provides some examples of enforced decomposition in flexible manufacturing and multiarmed bandit problems; see also the thesis by Kimemia [Kim82], and the papers by Kimemia, Gershwin, and Bertsekas [KGB82], and Whittle [Whi88]. The author's paper [Ber07] describes a few additional schemes based on constraint relaxation.

Section 2.4: The main idea of rollout algorithms, obtaining an improved policy starting from some other suboptimal policy, has appeared in several DP application contexts. The name "rollout" was coined by Tesauro in specific reference to rolling the dice in the game of backgammon [TeG96]. In Tesauro's proposal, a given backgammon position is evaluated by "rolling out" many games starting from that position, using a simulator, and the results are averaged to provide a "score" for the position; see Example 2.4.3. The use of the name "rollout" has gradually expanded beyond its original context; for example the samples collected through simulated trajectories are referred to as "rollouts" by some authors. In this book, we will adopt the original intended meaning: policy improvement starting from a base policy, which is evaluated with some form of Monte Carlo simulation.

The application of rollout algorithms to discrete deterministic optimization problems, the notions of sequential consistency, sequential improvement, fortified rollout, and the use of multiple heuristics (also called "parallel rollout") were first given in the paper by Bertsekas, Tsitsiklis, and Wu [BTW97], and also in the neuro-dynamic programming book by Bertsekas and Tsitsiklis [BeT96]. Rollout algorithms for stochastic problems were further formalized in the papers by Bertsekas [Ber97b], [Ber05a], [Ber05b], and Bertsekas and Castanon [BeC99]. A discussion of rollout algorithms, specialized to network optimization problems, was given in the

author's textbook [Ber98], Section 10.5.

The expert-based rollout scheme of Section 2.4.3 seems new in the form given here, but is inspired by the method of *comparison training* proposed by Tesauro [Tes89a], [Tes89b], [Tes01], and subsequently used by several other authors (for some recent references, see [DNW16], [TCW19]). This is a general method for training an approximation architecture to choose between two alternatives, using a dataset of expert choices in place of an explicit cost function.

There have been many works and applications relating to rollout algorithms. See Secomandi [Sec00], [Sec01], [Sec03], Ferris and Voelker [FeV02], [FeV04], McGovern, Moss, and Barto [MMB02], Savagaonkar, Givan, and Chong [SGC02], Bertsimas and Popescu [BeP03], Guerriero and Mancini [GuM03], Tu and Pattipati [TuP03], Wu, Chong, and Givan [WCG03], Chang, Givan, and Chong [CGC04], Meloni, Pacciarelli, and Pranzo [MPP04], Yan, Diaconis, Rusmevichientong, and Van Roy [YDR04], Besse and Chaib-draa [BeC08], Sun et al. [SZL08], Bertazzi et al. [BBG13], Sun et al. [SLJ13], Tesauro et al. [TGL13], Beyme and Leung [BeL14], Goodson, Thomas, and Ohlmann [GTO15], Li and Womer [LiW15], Mastin and Jaillet [MaJ15], Huang, Jia, and Guan [HJG16], Simroth, Holfeld, and Brunsch [SHB15], Lan, Guan, and Wu [LGW16], Ulmer [Ulm17], Bertazzi and Secomandi [BeS18], Guerriero, Di Puglia, and Macrina [GDP18], Ulmer at al. [UGM18], Chu, Xu, and Li [CXL19]. A recent survey of rollout in discrete optimization is given by the author in [Ber13a]. These works discuss variants and problem-specific adaptations of rollout algorithms for a broad variety of practical problems, and consistently report positive computational experience.

The idea of rollout that uses limited lookahead, adaptive pruning of the lookahead tree, and rollout truncation with cost function approximation at the end of the rollout was suggested by Tesauro and Galperin [TeG96] in the context of backgammon. Related ideas appeared earlier in the paper by Abramson [Abr90], in a game playing context.

The paper and the 2007 1st edition of the monograph by Chang, Hu, Fu, and Marcus [CFH05], [CFH13] proposed and analyzed adaptive sampling in connection with DP, and early forms of Monte Carlo tree search, including statistical tests to control the sampling process. The name "Monte Carlo tree search" (Section 2.4.2) has become popular, and in its current use, it encompasses a broad range of methods that involve adaptive sampling, rollout, extensions to sequential games, and the use and analysis of various statistical tests. We refer to the papers by Coulom [Cou06], the survey by Browne et al. [BPW12], and the discussions by Chang et al. [CFH16] and Fu [Fu17]. The development of statistical tests for adaptive sampling has been influenced by works on multiarmed bandit problems; see the papers by Lai and Robbins [LaR85], Agrawal [Agr95], Burnetas and Katehakis [BuK97], Meuleau and Bourgine [MeB99], Auer, Cesa-Bianchi, and Fischer [ACF02], Peret and Garcia [PeG04], Kocsis and Szepesvari [KoS06],

Dimitrakakis and Lagoudakis [DiL08], Audibert, Munos, and Szepesvari [AMS09], and the monograph by Munos [Mun14]. The technique for variance reduction in the calculation of Q-factor differences (Section 2.4.2) was given in the author's paper [Ber97b].

Rollout can be applied to problems with partial state information, reformulated in belief space. The computational requirements increase substantially over the perfect state information case, but not prohibitively so. We discuss some of the possibilities in Section 5.7.3, in the context of infinite horizon problems and policy iteration, including the use of rollout truncation, cost function approximation, and parallelization.

Section 2.5: The MPC approach is popular in a variety of control system design contexts, and particularly in chemical process control and robotics, where meeting explicit control and state constraints is an important practical issue. However, it requires a mathematical model and it is better suited for deterministic than for stochastic problems.

The literature on MPC is voluminous. For a survey, which gives many of the early references, see Morari and Lee [MoL99], and for a more recent survey, see Mayne [May14]. For related textbooks, see Maciejowski [Mac02], Camacho and Bordons [CaB04], Kouvaritakis and Cannon [KoC15], and Borelli, Bemporad, and Morari [BBM17].

The view of MPC as a rollout algorithm, first suggested in the author's review paper [Ber05a], provides a suboptimal control perspective and a connection with the RL methodology. The stability analysis given here is based on the work of Keerthi and Gilbert [KeG88]. The recent paper by Krener [Kre19] discusses methods to estimate the optimal cost function for use as terminal cost function approximation, aiming to achieve stabilization with MPC lookahead that is as small as possible. Generally, any problem approximation method that yields a "good" estimate of the optimal cost-to-go function is a promising candidate for terminal cost function approximation in the context of MPC and rollout. Proving stability of the overall scheme, however, depends on whether this cost function approximation satisfies a sequential improvement condition (sometimes also referred to as a "Lyapunov condition" in the control literature).

In our account of MPC, we have restricted ourselves to deterministic problems possibly involving tight state constraints as well as control constraints. Problems with stochastic uncertainty and state constraints are more challenging because of the difficulty of guaranteeing that the constraints are satisfied; see the survey by Mayne [May14] for a review of various approaches that have been used in this context. The textbook [Ber17], Section 6.4, describes MPC for problems with set membership uncertainty and state constraints, using target tube/reachability concepts, which originated in the author's PhD thesis and subsequent papers [Ber71], [Ber72], [BeR71], [BeR73]. Target tubes were also used subsequently in MPC and other contexts by several authors; see the surveys by Blanchini

[Bla99] and Mayne [May14]. Reachability for continuous-time games has been studied by Mitchell, Bayen, and Tomlin [MBT05]. For an alternative recent approach to reachability, which is based on the notion of Conditional Value-at-Risk (CVaR), see Chapman et al. [CLT19].

Problems with partial state information pose considerable challenges for MPC. One possible approach for problems with a continuous control space is to combine MPC with certainty equivalence, whereby states are replaced with estimates, as discussed in Section 2.3.2. For POMDP with finite state and control spaces, MPC cannot be applied, at least in the form given here. However, rollout is generally applicable and may provide a viable option for suboptimal control. We discuss this possibility in Section 5.7.3.