Reinforcement Learning and Optimal Control

by

Dimitri P. Bertsekas

Massachusetts Institute of Technology

Chapter 2
Approximation in Value Space

DRAFT

This is Chapter 2 of the draft textbook “Reinforcement Learning and Optimal Control.” The chapter represents “work in progress,” and it will be periodically updated. It more than likely contains errors (hopefully not serious ones). Furthermore, its references to the literature are incomplete. Your comments and suggestions to the author at dimitrib@mit.edu are welcome. The date of last revision is given below.

December 14, 2018

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 an exponential increase of the required computation and memory storage as the problem's size 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.

Approximation in Value and Policy Space

There are two general approaches for DP-based suboptimal control. The first is *approximation in value space*, where we approximate the optimal cost-to-go functions J_k with some other functions $\tilde{J}_k$. We then obtain a suboptimal policy from the DP equation where J_k is replaced by $\tilde{J}_k$. In other words, the control used at state x_k is obtained from the minimization

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

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, we will focus on cost function approximation, but we will occasionally digress to discuss direct Q-factor approximation.

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 we minimize over $\ell > 1$ stages with the future cots approximated by a function $\tilde{J}_{k+\ell}$. In our initial discussion of approximation in value space in Section 2.1, we will focus on one-step lookahead. However, there are straightforward extensions of the main ideas to the multi-step context. These extensions will be discussed in Section 2.2.

The major alternative to approximation in value space is *approximation in policy space*, whereby we select the policy by using optimization over a suitable restricted class of policies. An important advantage of this approach is that the computation of controls during on-line operation of the system is often much easier compared with the minimization (2.1). However, this advantage can also be gained by combining approximation in value space with a policy approximation in a two-stage scheme:

(a) Obtain the approximately optimal cost-to-go functions $\tilde{J}_k$, thereby defining a corresponding suboptimal policy $\tilde{\mu}_k$, $k = 0, \ldots, N-1$, via Eq. (2.1).

(b) Approximate $\tilde{\mu}_k$, $k = 0, \ldots, N-1$, using some form of regression and a training set consisting of a large sample of pairs $\big(x_k, \tilde{\mu}_k(x_k)\big)$.

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.

2.1 VARIANTS OF APPROXIMATION IN VALUE SPACE

There are two major issues in a value space approximation scheme:

(1) How to compute the lookahead functions $\tilde{J}_k$ that are involved in the minimization (2.1). We refer to this as *training*. There are quite a few training approaches. Several of them are discussed in this chapter, and more will be discussed in subsequent chapters.

(2) How to perform the minimization (2.1) and implement the suboptimal policy $\tilde{\mu}_k$. We refer to this as *control generation* or *control selection*. Again there are several exact and approximate methods for control selection, which will be discussed in the present and the subsequent chapters.

In this section we will provide a high level discussion of these issues.

2.1.1 Methods for Computing Approximations in Value Space

Regarding the computation of $\tilde{J}_k$, we will consider four main 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.

(b) *On-line approximate optimization (Section 2.4)*: These methods typically 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. *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 5)*: This is a special but rather sophisticated form of problem approximation. 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 approximate cost-to-go functions produced by any one of these methods.

Additional variations of the above methods are obtained when we use 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 (Sections 2.1.2 and 2.4.2). Figure 2.1.1 provides a schematic illustration.

2.1.2 Off-Line and On-Line Methods

In approximation in value space an important consideration is whether the approximate cost-to-go functions $\tilde{J}_k$ and the suboptimal control functions $\tilde{\mu}_k$, $k = 0, \ldots, N-1$, 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 the controls $\tilde{\mu}_k(x_k)$ are computed on-line, but the on-line or off-line computation of $\tilde{J}_k$ is an important design choice. We thus distinguish between:

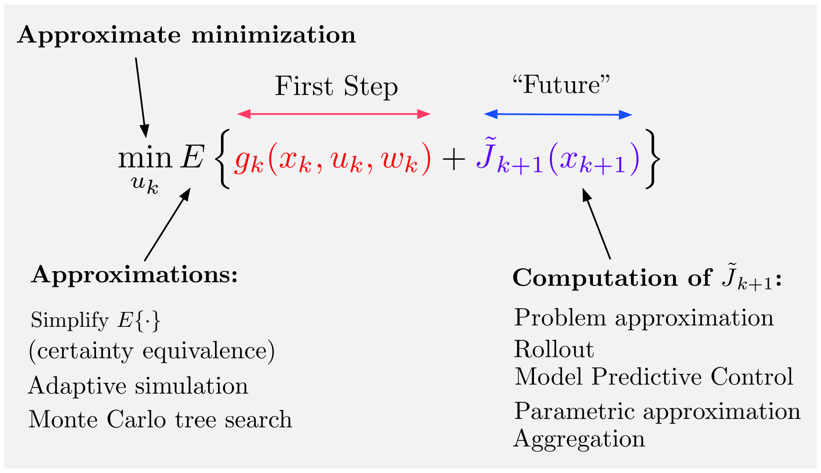

Figure 2.1.1 Schematic illustration of various options for approximation in value space with one-step lookahead. The lookahead function values $\tilde J_{k+1}(x_{k+1})$ approximate the optimal cost-to-go values $J_{k+1}(x_{k+1})$, and can be computed by a variety of methods. There may be additional approximations in the minimization over u_k and in the computation of the expected value over w_k; see Section 2.1.2.

(i) *Off-line methods*, where the entire function $\tilde J_k$ in Eq. (2.1) is computed for every k, before the control process begins. 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 the values $\tilde J_{k+1}(x_{k+1})$ 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 $\tilde J_k(x_{k+1})$ 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). These methods require the computation of a control only for the N states actually encountered in the control process. 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.

Of course there are also problem-dependent hybrid methods, where significant computation is done off-line to expedite the on-line computation of needed values of $\tilde J_k$.

2.1.3 Simplifying 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 in Eq. (2.1), once the cost-to-go approximating functions $\tilde J_k$ have been selected. An important concern here is the calculation of the expected value in the expression

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

as well as the minimization of this expression over $u_k \in U_k(x_k)$ [cf. Eq. (2.1)]. Both of these operations may involve substantial computation, which is of particular concern when the minimization is to be performed on-line. In this and subsequent sections, we will discuss several methods to alleviate this computation at the expense of additional approximation.

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 offers the potentially significant advantage that $\tilde J_k$ 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).

A related approach is to use *adaptive sampling*, whereby the expected value (2.2) is approximated with a few simulation samples, and based on the results of the approximation, some of the controls in $U_k(x_k)$ are discarded from further consideration. This approach may be used in stages, with progressively more accurate sampling, while discarding controls that appear to be nonpromising, and it may be based on sophisticated statistical analysis (confidence interval tests) for which we refer to the literature [CFH05], [Cou06], [KoS06], [CFH13], [Mun14], [Fu18]. Moreover progressive discarding of nonpromising controls can be used in a multistep lookahead setting, whereby for some state-control pairs (x_k,u_k) we use one-step lookahead, while for others we use multistep lookahead. In this context, adaptive sampling is also called *Monte-Carlo tree search*.

Finally, let us mention the issue of algorithmic minimization over $U_k(x_k)$ in Eqs. (2.1) 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, 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)].

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 may be attractive for deterministic problems, which lend themselves better to continuous space optimization; an example is the model predictive control context (see Section 2.4.3). For stochastic problems and either one-step or multistep lookahead and continuous control spaces, the methodology of *stochastic programming* may be useful. We refer to the textbook [Ber17] and the literature cited there. Still another possibility to simplify the one-step lookahead minimization (2.1) is based on Q-factor approximation, which is also suitable for model-free policy implementation, as we discuss next.

2.1.4 Model-Free Policy Implementation in Value and Policy Space

One of the major aims of this book is to address *model-free* situations, i.e., problems where a mathematical model [the system functions f_k, the probability distribution of w_k, and the one-stage cost functions g_k] is unavailable or hard to construct. We assume instead that the system and cost structure can be simulated far more easily (think, for example, of a queueing network with complicated but well-defined service disciplines at the queues). In particular, we assume that there is a computer program 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. We will now review some of the high-level ideas of passing from model-based to model-free policy implementations.

The minimization of Eq. (2.1) requires the functions f_k and g_k, and the probability distribution of w_k, so it is not model-free. To provide a model-free version, suppose that the values of $\tilde{J}_{k+1}(x_{k+1})$ and corresponding transition costs can be generated in a model-free fashion when needed (various approaches to obtain values of $\tilde{J}_{k+1}$ in model-free fashion will be discussed in the context of specific methods later; for example $\tilde{J}_{k+1}$ may be obtained by problem approximation using a simpler problem for which a model is available).

The idea is to 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. (2.1). The steps are as follows:

Summary of Model-Free Approximation in Value Space

(a) Use the simulator to collect a large number of "representative" sample state-control-disturbance triplets (x_k^s, u_k^s, w_k^s), successor states x_{k+1}^s, $s = 1, \ldots, q$, and corresponding sample Q-factors

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

(b) Determine the parameter vector r_k by the least-squares minimization

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

(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 model-free approximation procedure:

(1) Contrary to the minimization (2.1), 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 minimization (2.5) and the Q-factor minimization (2.6). The simulator to collect the samples (2.4) suffices.

(2) The policy $\tilde\mu_k$ obtained through the minimization (2.6) is not the same as the one obtained through the minimization (2.1). 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.

(3) 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.

Let us also mention a variant of the 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.

Model-Free Approximation in Policy 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 be 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

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

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

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

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

where $\tilde{J}_{k+1}$ is a suitable (separately obtained) approximation in value space; cf. Eq. (2.1), or an approximate Q-factor based minimization

$$u_k^s \in \arg \min_{u \in U_k(x_k)} \tilde{Q}_k(x_k^s, u, \bar{r}_k), \qquad i = 1, \ldots, n, \tag{2.9}$$

[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 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.

2.1.5 When is Approximation in Value Space Effective?

An important question is what constitutes good approximating functions $\tilde{J}_k$ in a one-step lookahead scheme. An answer that suggests itself is that $\tilde{J}_k$ should be "close" to the optimal cost-to-go function J_k for all k. This guarantees a certain degree of quality of the approximation scheme, but is neither necessary nor is it satisfied by all or even most good practical schemes.

For example if the approximating values $\tilde{J}_k(x_k)$ differ from the optimal values $J_k(x_k)$ uniformly by the same constant, the policy obtained by the approximation in value space scheme is optimal. This suggests that a better condition might be that relative values of $\tilde{J}_k$ and J_k should be "close" to each other, i.e.,

$$\tilde{J}_k(x_k) - \tilde{J}_{k+\ell}(x_k') \approx J_k(x_k) - J_k(x_k'),$$

for all pairs of states x_k and x_k'. Still, however, this guideline neglects the role of the first stage cost.

A more accurate predictor of good quality of the suboptimal policy obtained is that the Q-factor approximation error $Q_k(x_k, u) - \tilde{Q}_k(x_k, u)$ changes gradually as u changes, where Q_k and $\tilde{Q}_k$ denote the exactly optimal Q-factor and its approximation, respectively. For a heuristic explanation, suppose that approximation in value space generates a control $\tilde{u}_k$ at a state x_k where another control u_k is optimal. Then we have

$$\tilde{Q}_k(x_k, u_k) - \tilde{Q}_k(x_k, \tilde{u}_k) \geq 0, \tag{2.10}$$

since $\tilde{u}_k$ minimizes $\tilde{Q}_k(x_k, \cdot)$, and

$$Q_k(x_k, \tilde{u}_k) - Q_k(x_k, u_k) \geq 0, \tag{2.11}$$

since u_k minimizes $Q_k(x_k, \cdot)$. If $\tilde{u}_k$ is far from optimal, the Q-factor difference in Eq. (2.11) will be large, and by adding Eq. (2.10), it follows that the expression

$$\big(Q_k(x_k, \tilde{u}_k) - \tilde{Q}_k(x_k, \tilde{u}_k)\big) - \big(Q_k(x_k, u_k) - \tilde{Q}_k(x_k, u_k)\big)$$

will be even larger. This is not likely to happen if the approximation error $Q_k(x_k, u) - \tilde{Q}_k(x_k, u)$ changes gradually (i.e., has small "slope") for u in a neighborhood that includes u_k and $\tilde{u}_k$ (cf. Fig. 2.1.2). In many practical settings, the changes in the approximate Q-factors $\tilde{Q}_k(x_k, u)$ tend to be "similar " in form to the changes in the exact Q-factors $Q_k(x_k, u)$ as u changes, thus providing some explanation for the observed success of approximation in value space in practice.

Of course, one would like to have quantitative tests to check the quality of either the approximate cost functions $\tilde{J}_k$ and Q-factors $\tilde{Q}_k$, or the suboptimal policies obtained. However, general tests of this type are not available, and it is often hard to access how a particular suboptimal policy compares to the optimal, except on a heuristic, problem-dependent basis. Unfortunately, this is a recurring difficulty in approximate DP/RL.

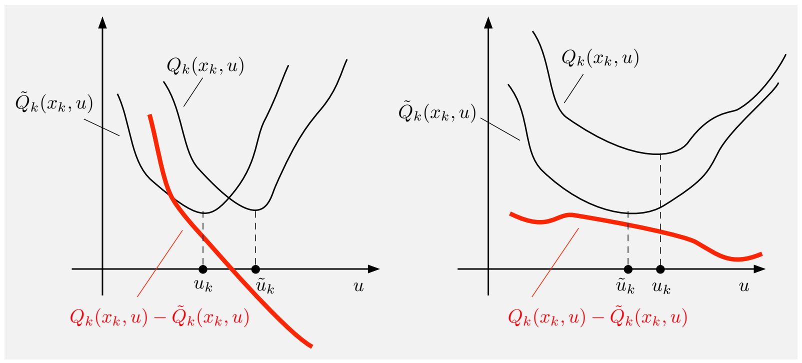

Figure 2.1.2 Schematic illustration of the "slope" of the Q-factor approximation error as a predictor of quality of an approximation in value space scheme. At a given state x_k, let u_k be optimal, so that it minimizes $Q_k(x_k,u)$ over $u \in U_k(x_k)$, and let $\tilde{u}_k$ be generated by approximation in value space, so that it minimizes $\tilde{Q}_k(x_k,u)$ over $u \in U_k(x_k)$. In the figure on the right the approximation error $Q_k(x_k,u) - \tilde{Q}_k(x_k,u)$ changes gradually (i.e., has small "slope"), and $\tilde{u}_k$ is a good choice, because $Q_k(x_k,\tilde{u}_k)$ is close the optimal $Q_k(x_k,u_k)$. In the figure on the left the approximation error $Q_k(x_k,u) - \tilde{Q}_k(x_k,u)$ changes fast (i.e., has large "slope"), and $\tilde{u}_k$ is a poor choice. In the extreme case where the Q-factors $Q_k(x_k,u)$ and $\tilde{Q}_k(x_k,u)$ differ by a constant, minimization of either one of them yields the same result.

2.2 MULTISTEP LOOKAHEAD

The approximation in value space scheme that we have discussed so far is known as one-step lookahead, since it involves solving a one-step minimization problem at each time k [cf. Eq. (2.1)]. A more ambitious, but also computationally more intensive scheme is *multistep lookahead*.

As an example, in *two-step lookahead* we apply at time k and state x_k, the control $\tilde{\mu}_k(x_k)$ attaining the minimum in Eq. (2.1), where now $\tilde{J}_{k+1}$ is obtained itself on the basis of a one-step lookahead approximation. In other words, for all possible states x_{k+1} that can be generated via the system equation starting from x_k,

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

we have

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

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

Thus two-step lookahead amounts to *solving a two-stage version of the DP problem with x_k as the initial state and $\tilde{J}_{k+2}$ as the terminal cost*

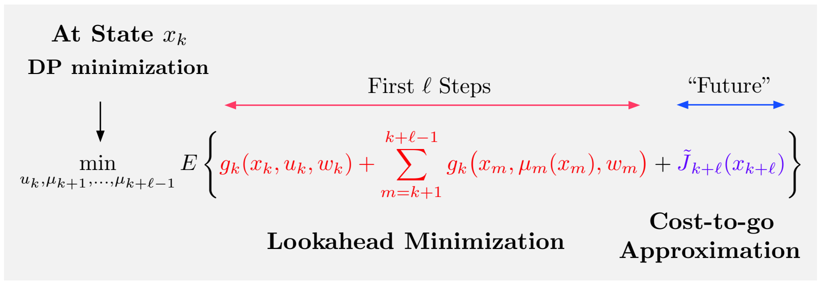
At State x_k
DP minimization
First ℓ Steps
"Future"
$\min_{u_k,\mu_{k+1},\ldots,\mu_{k+\ell-1}} E\left\{g_k(x_k,u_k,w_k)+\sum_{m=k+1}^{k+\ell-1} g_k\big(x_m,\mu_m(x_m),w_m\big)+\tilde J_{k+\ell}(x_{k+\ell})\right\}$
Lookahead Minimization
Cost-to-go Approximation

Figure 2.2.1 Schematic illustration of approximation in value space with ℓ-step lookahead.

function. Given x_k, the solution of this DP problem yields a two-stage policy that consists of the single control u_k for the first lookahead stage, plus a control $\mu_{k+1}(x_{k+1})$ for each value of $x_{k+1} = f_k(x_k, u_k, w_k)$ that can occur at the second lookahead stage $k + 1$. However, once this two-stage policy is computed, the controls $\mu_{k+1}(x_{k+1})$ are discarded, and only u_k is used as the control applied by the two-step lookahead policy at x_k. At the next stage, this process is repeated, i.e., we solve a two-stage DP problem with x_{k+1} as the initial state and $\tilde J_{k+3}$ as the terminal cost function.

Policies with lookahead of $\ell > 2$ stages are similarly defined: at state x_k, we solve an ℓ-stage version of the DP problem with x_k as the initial state and $\tilde J_{k+\ell}$ as the terminal cost function, and use the first control of the ℓ-stage policy thus obtained, while discarding the others; see Fig. 2.2.1. Of course, in the final stages where $k > N - \ell$, we should shorten the size of lookahead to $N - k$. Note that the simplifications in the one-step lookahead minimization discussed in Section 2.1.2 (assumed certainty equivalence, adaptive sampling, etc), and model-free policy implementation (Section 2.1.3) extend to multistep lookahead.

2.2.1 Multistep Lookahead and Rolling Horizon

There are several ways to compute the lookahead functions $\tilde J_k$ in multi-step lookahead, similar to the one-step lookahead case. However, there is also another possibility: with sufficiently long lookahead, we may capture enough of the character of the DP problem at hand so that a sophisticated choice of $\tilde J_k$ may not be needed.

In particular, we may set $\tilde J_{k+\ell}(x_{k+\ell}) \equiv 0$, or set $\tilde J_{k+\ell}(x_{k+\ell}) = g_N(x_{k+\ell})$. The idea is to use a sufficiently large number of lookahead steps ℓ to ensure a reasonably faithful approximation of the optimal cost-to-go functions $J_{k+\ell}$ within a constant.† This is also referred to as the *rolling*

† See the discussion in Section 2.1.4. Generally, rolling horizon schemes tend to work well if the probability distribution of the state $k+\ell$ steps ahead is roughly independent of the current state and control, or is concentrated around "low cost"

horizon approach, but essentially it is the same as multistep lookahead with a simplified cost-to-go approximation. Note that the idea of a rolling horizon is well-suited and applies with few modifications to infinite horizon problems as well.†

Typically, *as the size ℓ of the lookahead is chosen larger, the need for a good choice of $\tilde{J}_{k+\ell}$ diminishes*. One is tempted to conjecture that if ℓ is increased, then the performance of the lookahead policy is improved. This, however, need not be true always, essentially because beyond the next ℓ stages, the policy may be "blind" to the presence of particularly "favorable" or "unfavorable" states, as the following example illustrates.

Example 2.2.1

This is an oversimplified problem that demonstrates a basic pitfall of the multistep lookahead and rolling horizon approaches with approximate cost-to-go functions $\tilde{J}_k$ equal to 0. Consider a deterministic setting where at the initial state there are two possible controls, say 1 and 2 (see Fig. 2.2.2). At all other states there is only one control available, so a policy is specified by just the initial choice between controls 1 and 2.

Suppose that (based on the cost of the subsequent N stages) control 1 is optimal. Suppose also that if control 2 is chosen, an "unfavorable" (high cost) state results after ℓ stages, followed by a particularly "favorable" state, which is then followed by other "unfavorable" states. Then, in contrast with the ℓ-step lookahead policy, the $(\ell+1)$-step lookahead policy may view the inferior control 2 as being better, because it may be "fooled" by the presence of the particularly "favorable" state $\ell+1$ stages ahead.

2.2.2 Multistep Lookahead and Deterministic Problems

Generally, the implementation of multistep lookahead can be prohibitively time-consuming for stochastic problems, because it requires at each step the solution of a stochastic DP problem with a horizon that is equal to the size of the lookahead. However, when the problem is deterministic, the lookahead problems are also deterministic, and can be solved by shortest path methods for a finite spaces problem, or even for an infinite spaces problem after some form of discretization. This makes deterministic problems

states.

† For infinite horizon problems the cost-to-go approximations $\tilde{J}_k$ will typically be the same at all stages k, i.e., $\tilde{J}_k \equiv \tilde{J}$ for some $\tilde{J}$. As a result, the limited lookahead approach produces a stationary policy. In the case of discounted problems with an infinite horizon (see Chapter 4), a simple approach is to use a rolling horizon that is long enough so that the tail cost is negligible and can be replaced by zero, but it is also possible to use a small number of lookahead stages ℓ, as long as we compensate with a terminal cost function approximation $\tilde{J}$.

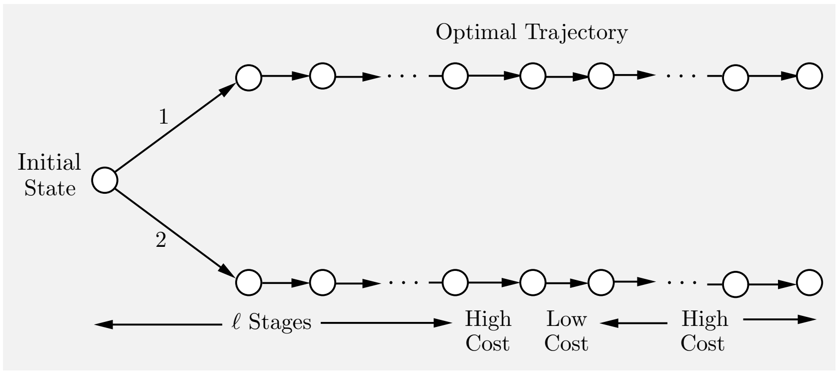

Figure 2.2.2 The deterministic problem of Example 2.2.1, which illustrates a basic pitfall of the multistep lookahead and rolling horizon approaches. Here, at the initial state there are two possible controls, 1 and 2. The control 1 is optimal, and is chosen if the comparison with control 2 is based on ℓ-step lookahead. However, control 2 is chosen if the comparison is based on $(\ell+1)$-step lookahead. Thus using a longer lookahead degrades the performance.

particularly good candidates for the use of long-step lookahead in conjunction with the rolling horizon approach that we discussed in the preceding section.

Even for a stochastic problem one may consider a hybrid approach that uses a partially deterministic form of one-step lookahead: at state x_k, allow for a stochastic disturbance w_k at the current stage, but fix the future disturbances $w_{k+1},\dots,w_{N-1}$ to some typical values, so as to bring to bear deterministic methods in the computation of approximate costs beyond the first stage.

In particular, with this approach, the needed values $\tilde J_{k+1}(x_{k+1})$ will be computed by deterministic shortest path methods, and will be used to compute the approximate Q-factors of pairs (x_k,u_k) using the formula

$$\tilde Q_k(x_k,u_k)=E\Big\{g_k(x_k,u_k,w_k)+\tilde J_{k+1}\big(f_k(x_k,u_k,w_k)\big)\Big\},$$

which incorporates the first stage uncertainty. Finally, the control chosen by such a scheme at time k will be

$$\tilde\mu_k(x_k)\in\arg\min_{u_k\in U_k(x_k)}\tilde Q_k(x_k,u_k);$$

cf. Eq. (2.6).

The idea of fixing uncertain quantities to typical values for approximation purposes is generally referred to as (assumed) *certainty equivalence*, and will be discussed at length in Section 2.3.2. The idea of using multistep lookahead for deterministic problems will also be reexamined in the context of the rollout algorithm in Section 2.4.1.

2.3 PROBLEM APPROXIMATION

A key issue in implementing a limited lookahead policy is the choice of the cost-to-go approximation at the end of the lookahead. In this section, we discuss the problem approximation approach whereby we approximate the optimal cost-to-go J_k with some function $\tilde{J}_k$ derived from a related but simpler problem (for example the optimal cost-to-go of that problem). In the following subsections we consider two approaches:

(1) *Simplifying the structure of the problem through enforced decomposition*, e.g., replacing coupling constraints with simpler decoupled constraints or with Lagrange multiplier-related penalties.

(2) *Simplifying the probabilistic structure of the problem*, e.g., replacing stochastic disturbances with deterministic ones.

Another approach that can be viewed as problem approximation is *aggregation*, whereby the original problem is approximated with a related "aggregate" problem that has smaller dimension or fewer states. This problem is solved exactly to yield a cost-to-go approximation for the original problem. This approach is also related to the feature-based parametric approximation ideas of Chapter 3, and will be discussed in Chapter 5.

2.3.1 Enforced Decomposition

The simplification/approximation approach is often well-suited for problems involving a number of subsystems that are coupled through the system equation, or the cost function, or the control constraints, but the degree of coupling is "relatively weak." It is difficult to define precisely what constitutes "weak coupling," but in specific problem contexts, usually this type of structure is easily recognized. For such problems it is often sensible to introduce approximations by artificially decoupling the subsystems in some way, thereby creating either a simpler problem or a simpler cost calculation, where subsystems can be dealt with in isolation.

There are a number of different ways to effect enforced decomposition, and the best approach is often problem-dependent. Generally, for a deterministic problem, enforced decomposition can be applied both off-line and on-line to produce a suboptimal control sequence. For a stochastic problem, it can be applied with off-line computation of the approximate cost-to-go functions $\tilde{J}_k$ and on-line computation of the corresponding suboptimal policy. We will illustrate these two possibilities with various application contexts in what follows.

Optimization of One Subsystem at a Time

When a problem involves multiple subsystems, a potentially interesting approximation approach is to optimize one subsystem at a time. In this way the control computation at time k may become simpler.

As an example consider an N-stage deterministic problem, where the control u_k at state x_k consists of n components, $u_k = \{u_k^1, \ldots, u_k^n\}$, with u_k^i corresponding to the ith subsystem. Then to compute a cost-to-go approximation at a given state x_k, one may optimize over the control sequence of a single subsystem, while keeping the controls of the remaining subsystems at some nominal values. Thus, upon arriving at x_k, we first optimize over the control sequence $\{u_k^1, u_{k+1}^1, \ldots, u_{N-1}^1\}$ of the first subsystem, then optimize over the controls of the second subsystem, and so on, while keeping the controls of the other subsystem at the latest "optimal" values computed.

There are several possible variations, for example to make the order in which the subsystems are considered subject to optimization as well, or to repeat cycling through the subsystems multiple times, each time using the results of the latest computation as nominal values of subsystem controls. This is similar to a "coordinate descent" approach, used in other optimization contexts.

Additional variations are obtained when we use approximate minimization over u_k in Eq. (2.1), and also when the expected value over w_k is computed approximately via adaptive simulation or a certainty equivalence approximation (cf. Section 2.1.2).

Example 2.3.1 (Vehicle Routing)

Consider n vehicles that move along the arcs of a given graph. Each node of the graph has a known "value" and the first vehicle that will pass through the node will collect its value, while vehicles that pass subsequently through the node do not collect any value. This may serve as a model of a situation where there are various valuable tasks to be performed at the nodes of a transportation network, and each task can be performed at most once and by a single vehicle. We assume that each vehicle starts at a given node and after at most a given number of arc moves, it must return to some other given node. The problem is to find a route for each vehicle satisfying these constraints, so that the total value collected by the vehicles is maximized.

This is a difficult combinatorial problem that in principle can be approached by DP. In particular, we can view as state the n-tuple of current positions of the vehicles together with the list of nodes that have been visited by some vehicle in the past, and have thus "lost" their value. Unfortunately, the number of these states is enormous (it increases exponentially with the number of nodes and the number of vehicles). The version of the problem that involves a single vehicle, while still difficult in principle, can often be solved in reasonable time either exactly by DP or fairly accurately using a suitable heuristic. Thus a one-step lookahead policy suggests itself, with the value-to-go approximation obtained by solving single vehicle problems.

In particular, in a one-step lookahead scheme, at a given time k and from a given state x_k we consider all possible n-tuples of moves by the n vehicles. At the resulting state x_{k+1} corresponding to each n-tuple of vehicle

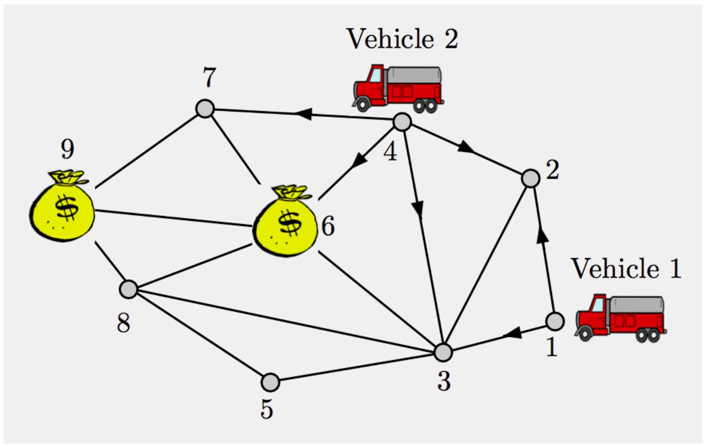

Figure 2.3.1 Schematic illustration of the vehicle routing problem and the one-vehicle-at-a-time approach. As an example, given the position pair $x_k = (1,4)$ of the two vehicles and the current valuable tasks at positions 6 and 9, we consider moves to all possible positions pairs x_{k+1}:

$$(2,2),\ (2,3),\ (2,6),\ (2,7),\ (3,2),\ (3,3),\ (3,6),\ (3,7).$$

From each of these pairs, we first compute the best route of vehicle 1 assuming vehicle 2 does not move, and then the best route vehicle 2, taking into account the previously computed route of vehicle 1. We then select the pair x_{k+1} that results in optimal value, and move the vehicles to the corresponding positions.

moves, we approximate the optimal value-to-go with the value corresponding to a suboptimal set of paths. These paths are obtained as follows: we fix an order of the vehicles and we calculate a path for the first vehicle, starting at x_{k+1}, assuming the other vehicles do not move. (This is done either optimally by DP, or near optimally using some heuristic.) Then we calculate a path for the second vehicle, taking into account the value collected by the first vehicle, and we similarly continue: for each vehicle, we calculate in the given order a path, taking into account the value collected by the preceding vehicles. We end up with a set of paths that have a certain total value associated with them. This is the value $\tilde J_{k+1}(x_{k+1})$ associated with the successor state x_{k+1}. We repeat with all successor states x_{k+1} corresponding to all the n-tuples of vehicle moves that are possible at x_k. We then use as suboptimal control at x_k the n-tuple of moves that yields the best value; see Fig. 2.3.1.

There are several enhancements and variations of the scheme just described. For example, we can consider several alternative orders for optimizing paths one-at-a-time, and choose the n-tuple of moves that corresponds to the best value obtained. Other variations may include travel costs between nodes of the graph, and constraints on how many tasks can be performed by each vehicle.

Constraint Decoupling by Constraint Relaxation

Let us now consider problems involving coupled subsystems where the coupling comes only through the control constraint. Typical cases involve the allocation of a limited resource to a set of subsystems whose system equations are completely decoupled from each other. We will illustrate with examples a few enforced decomposition approaches to deal with such situations. The first approach is *constraint relaxation*, whereby the constraint set is replaced by another constraint set that does not involve coupling.

Example 2.3.2 (Restless Multiarmed Bandit Problems)

An interesting DP model that arises in many application contexts involves n projects of which only one can be worked on at any time period. Each project i is characterized at time k by its state x_k^i. If project i is worked on at time k, one receives an expected reward $R^i(x_k^i)$, and the state x_k^i then evolves according to the equation

$$x_{k+1}^i = f^i(x_k^i, w_k^i), \qquad \text{if } i \text{ is worked on at time } k,$$

where w_k^i is a random disturbance with probability distribution depending on x_k^i but not on prior disturbances. If project i is not worked on, its state changes according to

$$x_{k+1}^i = \overline{f}^i(x_k^i, \overline{w}_k^i),$$

where $\overline{f}^i$ is a given function and $\overline{w}_k^i$ is a random disturbance with distribution depending on x_k^i but not on prior disturbances. Furthermore, a reward $\overline{R}^i(x_k^i)$ is earned, where $\overline{R}^i$ is a given function. The projects are coupled through the control constraint (only one of the projects may be worked on at any one period).† A suboptimal enforced decomposition approach is to consider the n single project problems where a single project is worked on through the entire remaining horizon, and add the contributions of the n problems to form an optimal reward approximation.

In particular, suppose that the optimal reward function $J_k(x^1, \ldots, x^n)$ is approximated by a separable function of the form

$$\sum_{i=1}^{n} \tilde{J}_k^i(x^i),$$

† In the simplest version of the problem, the state of a project that is not worked on remains unchanged and produces no reward, i.e.,

$$x_{k+1}^i = x_k^i, \qquad \overline{R}^i(x_k^i) = 0, \qquad \text{if } i \text{ is not worked on at time } k.$$

This is the classical *multiarmed bandit problem*, which has been studied extensively and admits optimal policies with a nice structure that can be computationally exploited; see e.g., [Ber12]. The term "restless" in the title of the present example, introduced in [Whi88], refers to the fact that the states of the projects that are not worked on may change.

where each $\tilde{J}_k^i$ is a function that quantifies the contribution of the ith project to the total reward. The corresponding one-step lookahead policy selects at time k the project i that maximizes

$$R^i(x^i) + \sum_{j\neq i} \overline{R}^j(x^j) + E\left\{\tilde{J}_{k+1}^i\big(f^i(x^i,w^i)\big)\right\} + \sum_{j\neq i} E\left\{\tilde{J}_{k+1}^j\big(\overline{f}^j(x^j,\overline{w}^j)\big)\right\},$$

which can also be written as

$$R^i(x^i) - \overline{R}^i(x^i) + E\left\{\tilde{J}_{k+1}^i\big(f^i(x^i,w^i)\big) - \tilde{J}_{k+1}^i\big(\overline{f}^i(x^i,\overline{w}^i)\big)\right\}$$
$$+ \sum_{j=1}^n \left\{\overline{R}^j(x^j) + E\big\{\tilde{J}_{k+1}^j\big(\overline{f}^j(x^j,\overline{w}^j)\big)\big\}\right\}.$$

Noting that the last term in the above expression does not depend on i, it follows that the one-step lookahead policy takes the form

$$\text{work on project } i \qquad \text{if} \qquad \tilde{m}_k^i(x^i) = \max_j \big\{\tilde{m}_k^j(x^j)\big\},$$

where for all i,

$$\tilde{m}_k^i(x^i) = R^i(x^i) - \overline{R}^i(x^i) + E\big\{\tilde{J}_{k+1}^i\big(f^i(x^i,w^i)\big) - \tilde{J}_{k+1}^i\big(\overline{f}^i(x^i,\overline{w}^i)\big)\big\}.$$

An important question for the implementation of the preceding suboptimal control scheme is the determination of the separable reward function terms $\tilde{J}_{k+1}^i$. There are several possibilities here, and the best choice may strongly depend on the problem's structure. One possibility is to compute $\tilde{J}_{k+1}^i$ as the optimal cost-to-go function for a problem involving just project i, i.e., assuming that none of the other projects $j \neq i$ will be worked on for the remaining periods $k+1,\ldots,N-1$. This corresponds to restricting the control constraint set of the problem, and involves a single-project optimization that may be tractable.

An alternative possibility is to use a separable parametric approximation of the form

$$\sum_{i=1}^n \tilde{J}_{k+1}^i(x_{k+1}^i, r_{k+1}^i),$$

where r_{k+1}^i are vectors of "tunable" parameters. The values of r_{k+1}^i can be obtained by some "training" algorithm such as the ones to be discussed in Chapter 3.

Constraint Decoupling by Lagrangian Relaxation

Another approach to deal with coupled constraints is to replace them with linear Lagrange multiplier-related penalty functions that are added to the cost function. We illustrate this approach with an extension of the preceding multiarmed bandit Example 2.3.2.

Example 2.3.3 (Separable Lower Bound Approximation in Multiarmed Bandit Problems)

Let us consider a version of the multiple projects Example 2.3.2 involving a more general form of control. Here there are n subsystems, and a control u_k^i is applied to subsystem i at time k. Instead of the requirement that only one subsystem is worked on at any one time, we assume a control constraint of the form

$$u_k = (u_k^1, \dots, u_k^n) \in U, \qquad k = 0, 1, \dots,$$

where the set U is given (Example 2.3.2 is obtained as the special case where U consists of the union of the coordinate vectors, i.e., those whose components are equal to 0, except for one component that is equal to 1). The ith subsystem is described by

$$x_{k+1}^i = f^i(x_k^i, u_k^i, w_k^i), \qquad i = 1, \dots, n,\ k = 0, 1, \dots,$$

where x_k^i is the state taking values in some space, u_k^i is the control, w_k^i is a random disturbance, and f^i is a given function. We assume that w_k^i is selected according to a probability distribution that may depend on x_k^i and u_k^i, but not on prior disturbances or the disturbances w_k^j of the other subsystems $j \neq i$. The cost incurred at the kth stage by the ith subsystem is

$$g^i(x_k^i, u_k^i, w_k^i), \tag{2.12}$$

where g^i is a given one-stage cost function. For notational convenience, we assume stationarity of the system equation and the cost per stage, but the approach to be discussed applies to the nonstationary case as well.

One possibility for a separable approximation of the problem is to replace the constraint $u_k \in U$ by a smaller or larger decoupled constraint, i.e., requiring that

$$u_k^i \in U^i, \qquad i = 1, \dots, n,\ k = 0, 1, \dots,$$

where the subsets $U^1, \dots, U^n$ satisfy $U^1 \times \cdots \times U^n \subset U$ or $U \subset U^1 \times \cdots \times U^n$, respectively.

We discuss another possibility for the case where the constraint set U includes linear inequality constraints. As a simple example, let us focus on a constraint set U of the form

$$U = \left\{ (u^1, \dots, u^n) \;\middle|\; u^i \in U^i \subset \Re,\ i = 1, \dots, n,\ \sum_{i=1}^n c^i u^i \leq b \right\}, \tag{2.13}$$

where $c^1, \dots, c^n$, and b are some scalars.† Here we replace the coupling constraint

$$\sum_{i=1}^n c^i u_k^i \leq b, \qquad k = 0, 1, \dots, \tag{2.14}$$

† More general cases, where u^i and b are multi-dimensional, and c^i are replaced by matrices of appropriate dimension, can be handled similarly, albeit with greater computational complications.

by a "relaxed" (larger) constraint whereby we require that

$$\sum_{k=0}^{N-1} \sum_{i=1}^{n} c^i u_k^i \leq Nb. \tag{2.15}$$

Roughly speaking, the constraint (2.15) requires that the coupling constraint (2.14) is satisfied "on the average," over the N stages.

We may now obtain a lower bound approximation of the optimal cost of our problem by assigning a scalar Lagrange multiplier $\lambda \geq 0$ to the constraint (2.15), and add a Lagrangian term

$$\lambda \left(\sum_{k=0}^{N-1} \sum_{i=1}^{n} c^i u_k^i - Nb \right) \tag{2.16}$$

to the cost function. This amounts to replacing the kth stage cost (2.12) by

$$g^i(x_k^i, u_k^i, w_k^i) + \lambda c^i u_k^i,$$

while replacing the coupling constraint (2.13) with the decoupled constraint

$$u_k^i \in U^i, \qquad i = 1, \ldots, n,$$

cf. Eq. (2.13). This is a lower bound approximation, as is typical in Lagrange multiplier-based decomposition approaches in linear and nonlinear programming (see e.g., [BeT97], [Ber16b]). To see this, note that for every feasible solution of the original problem, the Lagrangian term (2.16) makes a nonpositive contribution when added to the cost function, while with the constraint relaxed, the resulting optimal cost can only be reduced further.

With the subsystems now decoupled, we may solve each single subsystem problem separately, thereby obtaining a separable lower bound approximation

$$\sum_{i=1}^{n} \tilde{J}_k^i(x^i, \lambda)$$

for every $k = 1, \ldots, N-1$. This approximation can in turn be used to obtain a suboptimal one-step lookahead policy. Note that we may also try to optimize the approximation over λ, either by ad hoc experimentation or by a more systematic optimization method. † Another possibility is to use a more general Lagrangian term of the form

$$\left(\sum_{k=0}^{N-1} \lambda_k \sum_{i=1}^{n} c^i u_k^i - Nb \right),$$

in place of the term (2.16), where $\lambda_0, \ldots, \lambda_{N-1} \geq 0$ are time-varying scalar multipliers.

† Maximization of the lower bound approximation over λ is an interesting possibility. This is common in duality-based optimization. Generally the approach of this example falls within the framework of *Lagrangian relaxation*, a decomposition method that is based on the use of Lagrange multipliers and duality theory; see e.g., [BeT97], [Ber15a], [Ber16a].

2.3.2 Probabilistic Approximation - Certainty Equivalent Control

We will now consider problem approximation based on modifying the underlying probabilistic structure. The most common example of this type is the *certainty equivalent controller* (CEC). It replaces the stochastic disturbances with deterministic variables that are fixed at some "typical" values. Thus it acts as if a form of the certainty equivalence principle were holding, cf. the discussion of linear quadratic problems in Section 1.3.7.

The advantage of the CEC is that it involves a much less demanding computation than the stochastic DP algorithm: it requires the solution of a *deterministic* optimal control problem at each stage. This problem yields an optimal control sequence, the first component of which is used at the current stage, while the remaining components are discarded. Thus the CEC is able to deal with stochastic and even partial information problems by using the mature and more flexible methodology of deterministic optimal control.

We will describe the CEC for the stochastic DP problem of Section 1.2. Suppose that for every state-control pair (x_k, u_k) we have selected a "typical" value of the disturbance, which we denote by $\tilde{w}_k(x_k, u_k)$. For example the expected values

$$\tilde{w}_k(x_k, u_k) = E\{w_k \mid x_k, u_k\},$$

can serve as typical values, if the disturbance spaces are convex subsets of Euclidean spaces [so that they include $\tilde{w}_k(x_k, u_k)$].

To implement the CEC at state x_k and stage k we solve a deterministic optimal control problem obtained from the original problem by replacing all uncertain quantities by their typical values. In particular, we solve the problem

$$\min_{\substack{x_{i+1}=f_i(x_i,u_i,\tilde{w}_i(x_i,u_i))\\ u_i\in U_i(x_i),\ i=k,\ldots,N-1}} \left[g_N(x_N) + \sum_{i=k}^{N-1} g_i\big(x_i, u_i, \tilde{w}_i(x_i, u_i)\big) \right]. \tag{2.17}$$

If $\{\tilde{u}_k, \ldots, \tilde{u}_{N-1}\}$ is the optimal control sequence for this problem, we use the first control in this sequence and discard the remaining controls:

$$\tilde{\mu}_k(x_k) = \tilde{u}_k.$$

An alternative implementation of the CEC is to compute off-line an optimal policy

$$\big\{\mu_0^d(x_0), \ldots, \mu_{N-1}^d(x_{N-1})\big\}$$

for the deterministic problem

$$\begin{aligned} &\text{minimize} \ \ g_N(x_N) + \sum_{k=0}^{N-1} g_k\big(x_k, \mu_k(x_k), \tilde{w}_k(x_k, u_k)\big) \\ &\text{subject to} \ \ x_{k+1} = f_k\big(x_k, \mu_k(x_k), \tilde{w}_k(x_k, u_k)\big), \quad \mu_k(x_k) \in U_k, \quad k \geq 0, \end{aligned} \tag{2.18}$$

by using the DP algorithm. Then the control input $\tilde{\mu}_k(I_k)$ applied by the CEC at time k is given by

$$\tilde{\mu}_k(I_k) = \mu_k^d(x_k).$$

The two variants of the CEC just given are equivalent in terms of performance. The main difference is that the first variant is well-suited for on-line replanning, while the second variant is more suitable for an off-line implementation.

Finally, let us note that the CEC can be extended to imperfect state observation problems, where the state x_k is not known at time k, but instead an estimate of x_k is available, which is based on measurements that have been obtained up to time x_k. In this case, we find a suboptimal control similarly, as in Eqs. (2.17) and (2.18), but with x_k replaced by the estimate, as if this estimate were exact.

Certainty Equivalent Control with Heuristics

Even though the CEC approach simplifies a great deal the computations, it still requires the solution of a deterministic optimal control tail subproblem at each stage [cf. Eq. (2.17)], or the solution by DP of an N-stage deterministic optimal control problem [cf. Eq. (2.18)]. In the former case, these problems may still be difficult, and a more convenient approach may be to solve them suboptimally using a heuristic algorithm. In particular, given the state x_k at time k, we may use some (easily implementable) heuristic to find a suboptimal control sequence $\{\tilde{u}_k, \tilde{u}_{k+1}, \ldots, \tilde{u}_{N-1}\}$ for the problem of Eq. (2.17), and then use $\tilde{u}_k$ as the control for stage k.

An important enhancement of this idea is to use minimization over the first control u_k and to use the heuristic only for the remaining stages $k+1, \ldots, N-1$. To implement this variant of the CEC, we must apply at time k a control $\tilde{u}_k$ that minimizes over $u_k \in U_k(x_k)$ the expression

$$g_k\big(x_k, u_k, \tilde{w}_k(x_k, u_k)\big) + H_{k+1}(x_{k+1}), \tag{2.19}$$

where

$$x_{k+1} = f_k\big(x_k, u_k, \tilde{w}_k(x_k, u_k)\big), \tag{2.20}$$

and H_{k+1} is the cost-to-go function corresponding to the heuristic, i.e., $H_{k+1}(x_{k+1})$ is the cost incurred over the remaining stages $k+1, \ldots, N-1$ starting from a state x_{k+1}, using the heuristic. This is a hybrid approach: it resembles one-step lookahead with lookahead function H_{k+1}, and it resembles certainty equivalence in that the uncertain quantities have been replaced by their typical values.

Note that for any next state x_{k+1}, it is not necessary to have a closed-form expression for the heuristic cost-to-go $H_{k+1}(x_{k+1})$. Instead we can generate this cost by running the heuristic from x_{k+1} and computing the corresponding cost. Thus all the possible next states x_{k+1} must be computed for all possible values of the control u_k, and then the heuristic must be run from each of these x_{k+1} to calculate $H_{k+1}(x_{k+1})$, which is needed in the minimization of the expression (2.19).

Example 2.3.4 (Parking)

Consider the parking problem of Example 1.3.3, where a driver is looking for a parking space in a parking area consisting of N spaces arranged in a line, with a garage at the end of the line. The driver starts at space 0 and traverses the parking spaces sequentially, i.e., from each space he goes to the next space, up to when he decides to park in space k at cost $c(k)$, if space k is free, or upon reaching the garage where parking is mandatory at cost C. In Example 1.3.3, we assumed that space k is free with a fixed probability $p(k) \equiv p$ independently of whether other parking spaces are free or not.

Assume now instead that $p(k)$ is an estimate that is based on the driver's observations of the status of preceding spaces $k+1, \ldots, N$. For example, this estimation process may involve exploiting correlations that may exist between the parking statuses of different spaces. Thus we assume that the driver, upon arrival at space k, knows the belief state of the spaces that lie ahead, i.e., the vector of conditional probabilities $\big(p(k-1), \ldots, p(0)\big)$ given the observations of the spaces $k+1, \ldots, N$.

The problem now becomes very hard to solve by exact DP, because the state space is infinite: at time k the state consists of the free/taken status of the current position k, plus the belief state of the spaces that lie ahead. However, a simple suboptimal approach to the problem can be based on certainty equivalence: at time k, we fix the free/taken probabilities of the spaces that lie ahead to their current belief values, and act as if these values will not change in the future. Then upon reaching space k, the fast DP algorithm of Example 1.3.3 can be used to solve on-line the corresponding fixed probabilities problem, and to find the corresponding suboptimal decision.

Partial Certainty Equivalent Control

In the preceding descriptions of the CEC all future and present uncertain quantities are fixed at their typical values. A useful variation is to fix at typical values only *some* of these quantities. For example, a partial state information problem may be treated as one of perfect state information, using an estimate $\tilde{x}_k$ of x_k as if it were exact, while fully taking into account the stochastic nature of the disturbances. Thus, if $\big\{\mu_0^p(x_0), \ldots, \mu_{N-1}^p(x_{N-1})\big\}$ is an optimal policy obtained from the DP algorithm for the stochastic perfect state information problem

$$\text{minimize} \ \ E\left\{ g_N(x_N) + \sum_{k=0}^{N-1} g_k\big(x_k, \mu_k(x_k), w_k\big) \right\}$$

$$\text{subject to} \ \ x_{k+1} = f_k\big(x_k, \mu_k(x_k), w_k\big), \quad \mu_k(x_k) \in U_k, \quad k = 0, \ldots, N-1,$$

then the control input applied by this variant of CEC at time k is $\mu_k^p(\tilde{x}_k)$, where $\tilde{x}_k$ is the estimate of x_k given the information available up to time k. Let us provide an example.

Example 2.3.5 (The Unscrupulous Innkeeper)

Consider an innkeeper who charges one of m different rates $r_1, \ldots, r_m$ for a room as the day progresses, depending on whether he has many or few vacancies, so as to maximize his expected total income during the day. A quote of a rate r_i is accepted with probability p_i and is rejected with probability $1 - p_i$, in which case the customer departs, never to return during that day. When the number y of customers that will ask for a room during the rest of the day (including the customer currently asking for a room) is known and the number of vacancies is x, the optimal expected income $\tilde{J}(x, y)$ of the innkeeper is given by the DP algorithm

$$\tilde{J}(x,y) = \max_{i=1,\ldots,m} \Big[p_i\big(r_i + \tilde{J}(x-1, y-1)\big) + (1-p_i)\tilde{J}(x, y-1)\Big], \tag{2.21}$$

for all $x \geq 1$ and $y \geq 1$, with initial conditions

$$\tilde{J}(x,0) = \tilde{J}(0,y) = 0, \qquad \text{for all } x \text{ and } y.$$

This algorithm can be used to obtain the values of $\tilde{J}(x, y)$ for all pairs (x, y).

Consider now the case where the innkeeper does not know y at the times of decision, but instead only maintains a probability distribution for y. Then, it can be seen that the problem becomes a difficult partial state information problem. The exact DP algorithm should then be executed over the set of the pairs of x and the belief state. Yet a reasonable partially stochastic CEC is based on approximating the optimal cost-to-go of subsequent decisions with $\tilde{J}(x-1, \tilde{y}-1)$ or $\tilde{J}(x, \tilde{y}-1)$, where the function $\tilde{J}$ is calculated by the preceding recursion (2.21) and $\tilde{y}$ is an estimate of y, such as the closest integer to the expected value of y. In particular, according to this one-step lookahead policy, when the innkeeper has a number of vacancies $x \geq 1$, he quotes to the current customer the rate that maximizes

$$p_i\big(r_i + \tilde{J}(x-1, \tilde{y}-1) - \tilde{J}(x, \tilde{y}-1)\big).$$

Thus in this suboptimal scheme, the innkeeper acts as if the estimate $\tilde{y}$ were exact.

Other Variations of Certainty Equivalent Control

It is also possible to use more general approaches to implement one-step lookahead control based on simplification of the probabilistic structure of the problem. The possibilities are highly problem-dependent by nature, but we may distinguish a few general techniques, which we illustrate through examples.

Example 2.3.6 (Decoupling Disturbance Distributions)

Let us consider a CEC approach in the context of enforced decomposition (cf. Section 2.3.1) when the subsystems are coupled only through the disturbance. In particular, consider n subsystems of the form

$$x_{k+1}^i = f^i(x_k^i, u_k^i, w_k^i), \qquad i = 1, \ldots, n.$$

Here the ith subsystem has its own state x_k^i, control u_k^i, and cost per stage $g^i(x_k^i, u_k^i, w_k^i)$, but the probability distribution of w_k^i depends on the full state $x_k = (x_k^1, \ldots, x_k^n)$.

A natural form of suboptimal control is to solve at each stage k and for each i, the ith subsystem optimization problem where *the probability distribution of the future disturbances* $w_{k+1}^i, \ldots, w_{N-1}^i$ *is "decoupled,"* in the sense that it depends only on the corresponding "local" states $x_{k+1}^i, \ldots, x_{N-1}^i$. This distribution may be derived by using some nominal values $\tilde{x}_{k+1}^j, \ldots, \tilde{x}_{N-1}^j$, $j \neq i$, of the future states of the other subsystems, and these nominal values may in turn depend on the full current state x_k. The first control $\overline{u}_k^i$ in the optimal policy thus obtained is applied at the ith subsystem in stage k, and the remaining portion of this policy is discarded.

Example 2.3.7 (Approximation Using Scenarios)

We noted earlier the possibility to approximate the optimal cost-to-go with a CEC approach, whereby for a given state x_{k+1} at time $k+1$, we fix the remaining disturbances at some nominal values $\tilde{w}_{k+1}, \ldots, \tilde{w}_{N-1}$, and we compute an optimal control or heuristic-based trajectory starting from x_{k+1} at time $k+1$.

This CEC approximation involves a single nominal trajectory of the remaining uncertainty. To strengthen this approach, it is natural to consider multiple trajectories of the uncertainty, called *scenarios*, and to construct an approximation to the optimal cost-to-go that involves, for every one of the scenarios, the cost of either an optimal or a heuristic policy.

Mathematically, we assume that we have a method, which at a given state x_{k+1}, generates q uncertainty sequences

$$w^s(x_{k+1}) = (w_{k+1}^s, \ldots, w_{N-1}^s), \qquad s = 1, \ldots, q.$$

These are the scenarios considered at state x_{k+1}. The cost $J_{k+1}(x_{k+1})$ is then approximated by

$$\tilde{J}_{k+1}(x_{k+1}, r) = \sum_{s=1}^{q} r_s C_s(x_{k+1}), \tag{2.22}$$

where $r = (r_1, \ldots, r_q)$ is a probability distribution, i.e., a vector of nonnegative parameters that add to 1, and $C_s(x_{k+1})$ is the cost corresponding to an occurrence of the scenario $w^s(x_{k+1})$, when starting from state x_{k+1} and using either an optimal or a given heuristic policy.

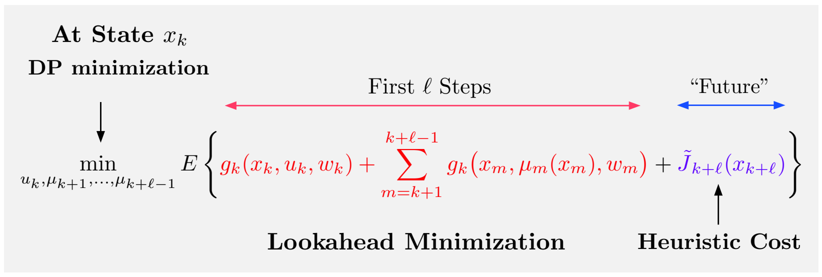
At State x_k
DP minimization
First ℓ Steps
"Future"
$\min_{u_k,\mu_{k+1},\ldots,\mu_{k+\ell-1}} E\left\{g_k(x_k,u_k,w_k)+\sum_{m=k+1}^{k+\ell-1} g_k\big(x_m,\mu_m(x_m),w_m\big)+\tilde J_{k+\ell}(x_{k+\ell})\right\}$
Lookahead Minimization
Heuristic Cost

Figure 2.4.1 Schematic illustration of rollout with ℓ-step lookahead. The approximate cost $\tilde J_{k+\ell}(x_{k+\ell})$ is obtained by running a heuristic algorithm from state $x_{k+\ell}$.

The parameters $r_1,\ldots,r_q$ may depend on the time index, and may be interpreted as "aggregate probabilities" that encode the aggregate effect on the cost-to-go function of uncertainty sequences that are similar to the scenario $w^s(x_{k+1})$. They may be computed using some ad hoc scheme, or some more systematic approach. This idea is also related to the rollout approach that is the subject of the next section.

2.4 ROLLOUT AND MODEL PREDICTIVE CONTROL

Rollout in its purest form can be defined very simply: it is approximation in value space with the approximate cost-to-go values $\tilde J_{k+1}(x_{k+1})$ calculated by running some heuristic/suboptimal policy starting from each possible next state x_{k+1} (see Fig. 2.4.1). It can be combined with one-step or multistep lookahead. Moreover, in some cases involving a long horizon, the heuristic may be "truncated" in rolling horizon fashion, i.e., it may be run for a limited number of steps, with some some cost function approximation at the end (cf. Section 2.2).

We will describe rollout first for deterministic problems and one-step lookahead, and then for stochastic problems, in Sections 2.4.1 and 2.4.2, respectively. In Section 2.4.3, we discuss model predictive control, a dominant control system design methodology at present, which is a special case of rollout.

In Section 2.4.2, we will also discuss the use of rollout in an enhanced implementation of any given suboptimal policy, possibly obtained through some other approximation scheme. This is to use one-step or multistep lookahead with the cost function approximation consisting of two parts:

(a) Rollout with the given policy over a limited horizon.

(b) A terminal cost function approximation at the end of the rollout horizon, such as an estimate of the true cost function of the policy.

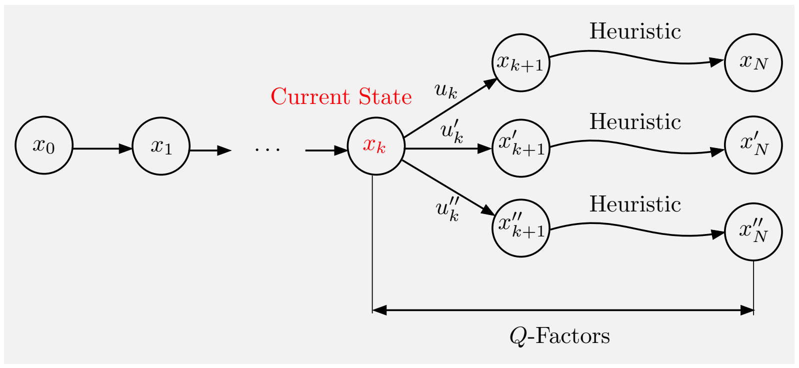

Figure 2.4.2 Schematic illustration of rollout with one-step lookahead for a deterministic problem. At state x_k, for every pair (x_k, u_k), $u_k \in U_k(x_k)$, the base heuristic generates a Q-factor $\tilde{Q}_k(x_k, u_k)$, and selects the control $\tilde{\mu}_k(x_k)$ with minimal Q-factor.

We will also discuss schemes of this type in Chapter 4, Section 4.5.3, in the context of infinite horizon problems.

2.4.1 Rollout for Deterministic Problems

Given a deterministic DP problem and any state x_k at time k, rollout uses some algorithm, called *base heuristic*, to generate a sequence of controls that solve suboptimally all the tail subproblems that start at every possible next state x_{k+1}. Thus when at x_k, the base heuristic, generates on-line for every $u_k \in U_k(x_k)$ the deterministic sequences of states $\{x_{k+1}, \ldots, x_N\}$ and controls $\{u_{k+1}, \ldots, u_{N-1}\}$ such that

$$x_{i+1} = f_i(x_i, u_i), \qquad i = k, \ldots, N-1.$$

Then the rollout algorithm applies the control that minimizes over $u_k \in U_k(x_k)$ the cost expression for stages k to N:

$$g_k(x_k, u_k) + g_{k+1}(x_{k+1}, u_{k+1}) + \cdots + g_{N-1}(x_{N-1}, u_{N-1}) + g_N(x_N). \quad (2.23)$$

More succinctly, the rollout algorithm applies at state x_k the control $\tilde{\mu}_k(x_k)$ given by the minimization

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

where $\tilde{Q}_k(x_k, u_k)$ is the approximate Q-factor defined by

$$\tilde{Q}_k(x_k, u_k) = g_k(x_k, u_k) + H_{k+1}\big(f_k(x_k, u_k)\big), \quad (2.24)$$

with $H_{k+1}(x_{k+1})$ denoting the cost of the base heuristic starting from state x_{k+1} [i.e., $H_{k+1}(x_{k+1})$ is the sum of all the terms in Eq. (2.23), except the first]; see Fig. 2.4.2. The rollout process defines a suboptimal policy $\tilde{\pi} = \{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\}$, referred to as the *rollout policy*.

Example 2.4.1 (Traveling Salesman Problem)

Let us consider the traveling salesman problem, whereby a salesman wants to find a minimum mileage/cost tour that visits each of N given cities $c = 0, \ldots, N-1$ exactly once and returns to the city he started from (cf. Example 2.3.2). With each pair of distinct cities c, c', we associate a traversal cost $a_{cc'}$. Note that we assume that we can go directly from every city to every other city. There is no loss of generality in doing so because we can assign a very high cost $a_{cc'}$ to any pair of cities (c, c') that is precluded from participation in the solution. The problem is to find a visit order that goes through each city exactly once and whose sum of costs is minimum.

There are many heuristic approaches for solving the traveling salesman problem. For illustration purposes, let us focus on the simple *nearest neighbor* heuristic. Here, we start from a path consisting of just a single city c_0 and at each iteration, we enlarge the path with a city that does not close a cycle and minimizes the cost of the enlargement. In particular, after k iterations, we have a sequence $\{c_0, \ldots, c_k\}$ consisting of distinct cities, and at the next iteration, we add a new city c_{k+1} that minimizes $a_{c_k c_{k+1}}$ over all cities $c_{k+1} \neq c_0, \ldots, c_k$. After the nearest neighbor heuristic selects city c_N, a complete tour is formed with total cost

$$a_{c_0 c_1} + \cdots + a_{c_{N-2} c_{N-1}} + a_{c_{N-1} c_0}. \tag{2.25}$$

We can formulate the traveling salesman problem as a DP problem as discussed in Example 1.3.1. We choose a starting city, say c_0, as the initial state x_0. Each state x_k corresponds to a city sequence/partial tour $(c_0, c_1, \ldots, c_k)$ consisting of distinct cities. The state x_{k+1}, next to x_k, are sequences of the form $(c_0, c_1, \ldots, c_k, c_{k+1})$ which correspond to adding one more unvisited city $c_{k+1} \neq c_0, c_1, \ldots, c_k$. The terminal states x_N are the complete tours of the form $(c_0, c_1, \ldots, c_{N-1}, c_0)$, and the cost of the corresponding sequence of city choices is the cost of the corresponding complete tour. Thus a state trajectory yields a complete tour and its total cost is the cost of the tour (2.25).

Let us now use as a base heuristic the nearest neighbor method. The corresponding rollout algorithm operates as follows: After $k < N$ iterations, we have a state x_k, i.e., sequence $\{c_0, \ldots, c_k\}$ consisting of distinct cities. At the next iteration, we add one more city as follows: We run the nearest neighbor heuristic starting from each of the sequences of the form $\{c_0, \ldots, c_k, c\}$ where $c \neq c_1, \ldots, c_k$. We then select as next city c_{k+1} the city c that yielded the minimum cost tour under the nearest neighbor heuristic.

Cost Improvement with a Rollout Algorithm - Sequential Consistency

The definition of the rollout algorithm leaves open the choice of the base heuristic. There are several types of common heuristics that can be used, such as greedy algorithms, local search, genetic algorithms, tabu search, and others. Clearly we want to choose a base heuristic that strikes a good balance between quality of solutions produced and computational tractability. However, some special conditions must hold in order to guarantee that the rollout policy has no worse performance than the base heuristic. We provide two such conditions, *sequential consistency* and *sequential improvement*, and then show how to modify the algorithm to deal with the case where these conditions are not satisfied.

We say that the base heuristic is *sequentially consistent* if it has the property that when it generates the sequence $\{x_k, x_{k+1}, \ldots, x_N\}$ starting from state x_k, it also generates the sequence $\{x_{k+1}, \ldots, x_N\}$ starting from state x_{k+1}. In other words, the base heuristic is sequentially consistent if it "stays the course": when the starting state x_k is moved forward to the next state x_{k+1} of its state trajectory, the heuristic will not deviate from the remainder of the trajectory.

As an example, the reader may verify that the nearest neighbor heuristic described in the traveling salesman Example 2.4.1 is sequentially consistent. Similar examples include the use of many types of greedy heuristics (see [Ber17], Section 6.4).

A simple condition that is equivalent to sequential consistency is that there exists a policy $\{\mu_0, \ldots, \mu_{N-1}\}$ such that the sequence generated by the base heuristic starting from any state x_k is the same as the one generated by the policy starting from the same state x_k. To see this, note that a policy clearly has the sequential consistency property, and conversely, a sequentially consistent base heuristic defines a policy: the one that moves from x_k to the state x_{k+1} that lies on the path $\{x_k, x_{k+1}, \ldots, x_N\}$ generated by the base heuristic.

Based on this fact, we can show that *the rollout algorithm obtained with a sequentially consistent base heuristic yields an improved cost over the base heuristic*. In particular, let us consider the rollout policy $\tilde{\pi} = \{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\}$, and let $J_{k,\tilde{\pi}}(x_k)$ denote the cost obtained with the base policy starting from state x_k. We claim that

$$J_{k,\tilde{\pi}}(x_k) \le H_k(x_k), \qquad \text{for all } x_k \text{ and } k, \tag{2.26}$$

where $H_k(x_k)$ denotes the cost of the heuristic starting from x_k.

We prove this inequality by induction. Clearly it holds for $k = N$, since $J_{N,\tilde{\pi}} = H_N = g_N$. Assume that it holds for index $k+1$. For any state x_k, let $\overline{u}_k$ be the control applied by the base heuristic at x_k. Then we have

$$\begin{aligned} J_{k,\tilde{\pi}}(x_k) &= g_k\big(x_k, \tilde{\mu}_k(x_k)\big) + J_{k+1,\tilde{\pi}}\Big(f_k\big(x_k, \tilde{\mu}_k(x_k)\big)\Big) \\ &\le g_k\big(x_k, \tilde{\mu}_k(x_k)\big) + H_{k+1}\big(f_k(x_k, \tilde{\mu}_k(x_k))\big) \\ &= \min_{u_k \in U_k(x_k)} \Big[g_k(x_k, u_k) + H_{k+1}\big(f_k(x_k, u_k)\big)\Big] \\ &\le g_k\big(x_k, \overline{u}_k\big) + H_{k+1}\big(f_k(x_k, \overline{u}_k)\big) \\ &= H_k(x_k), \end{aligned} \tag{2.27}$$

where:

(a) The first equality is the DP equation for the rollout policy $\tilde{\pi}$.

(b) The first inequality holds by the induction hypothesis.

(c) The second equality holds by the definition of the rollout algorithm.

(d) The third equality is the DP equation for the policy that corresponds to the base heuristic.

The induction proof of the cost improvement property (2.26) is thus complete.

Sequential Improvement

We will now show that the rollout policy has no worse performance than its base heuristic under a condition that is weaker than sequential consistency. Let us recall that the rollout algorithm $\tilde{\pi} = \{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\}$ is defined by the minimization

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

where $\tilde{Q}_k(x_k, u_k)$ is the approximate Q-factor defined by

$$\tilde{Q}_k(x_k, u_k) = g_k(x_k, u_k) + H_{k+1}\big(f_k(x_k, u_k)\big),$$

[cf. Eqs. (2.24)], and $H_{k+1}(x_{k+1})$ denotes the cost of the base heuristic starting from state x_{k+1}.

Let us assume that for all x_k and k, we have

$$\min_{u_k \in U_k(x_k)} \Big[g_k(x_k, u_k) + H_{k+1}\big(f_k(x_k, u_k)\big) \Big] \leq H_k(x_k). \tag{2.28}$$

Then, from the calculation of Eq. (2.27), skipping the second inequality which relies on sequential consistency, and using instead Eq. (2.28), we have

$$J_{k,\tilde{\pi}}(x_k) \leq H_k(x_k), \qquad \text{for all } x_k \text{ and } k.$$

If the base heuristic has the property (2.28) we say that it is *sequentially improving*. Thus the rollout algorithm obtained with a sequentially improving base heuristic, will improve or at least will perform no worse than the base heuristic, for every starting state x_k. Note that when the heuristic is sequentially consistent it is also sequentially improving, since in this case Eq. (2.28) is satisfied with equality.

Empirically, it has been observed that the cost improvement is typically considerable and often dramatic. Generally, however, it is hard to provide solid theoretical support for this observation. The price for the performance improvement is extra computation that is typically equal to the computation time of the base heuristic times a factor that is a low order polynomial of the problem size. Several case studies (see the end of chapter references) support this assessment. The textbook [Ber17], Section 6.4, provides some detailed worked-out examples.

On the other hand the sequential improvement condition may not hold for a given base heuristic. It is thus important to know that *any heuristic can be made to be sequentially improving with a simple modification*, as we explain next.

The Fortified Rollout Algorithm

We will describe a variant of the rollout algorithm that implicitly uses a sequentially improving base heuristic, so that it has the sequential improvement property (2.28). This variant, called the *fortified rollout algorithm*, starts at x_0, and generates step-by-step a sequence of states $\{x_0, x_1, \ldots, x_N\}$ and corresponding sequence of controls. Upon reaching state x_k it stores the trajectory

$$\{x_0, u_0, \ldots, u_{k-1}, x_k\}$$

that has been constructed up to stage k, called *permanent* trajectory, and it also stores a *tentative* trajectory

$$\overline{T}_k = \{x_k, \overline{u}_k, \overline{x}_{k+1}, \overline{u}_{k+1}, \ldots, \overline{u}_{N-1}, \overline{x}_N\}$$

with corresponding cost

$$C(\overline{T}_k) = g_k(x_k, \overline{u}_k) + g_{k+1}(\overline{x}_{k+1}, \overline{u}_{k+1}) + \cdots + g_{N-1}(\overline{x}_{N-1}, \overline{u}_{N-1}) + g_N(\overline{x}_N).$$

Initially, $\overline{T}_0$ is the trajectory generated by the base heuristic starting at the initial state x_0. The idea now is to deviate from the rollout algorithm at every state x_k where the base heuristic produces a trajectory with larger cost than $\overline{T}_k$, and use $\overline{T}_k$ instead.

In particular, upon reaching state x_k, we run the rollout algorithm as earlier, i.e., for every $u_k \in U_k(x_k)$ and next state $x_{k+1} = f_k(x_k, u_k)$, we run the base heuristic from x_{k+1}, and find the control $\tilde{u}_k$ that gives the best trajectory, denoted

$$\tilde{T}_k = \{x_k, \tilde{u}_k, \tilde{x}_{k+1}, \tilde{u}_{k+1}, \ldots, \tilde{u}_{N-1}, \tilde{x}_N\}$$

with corresponding cost

$$C(\tilde{T}_k) = g_k(x_k, \tilde{u}_k) + g_{k+1}(\tilde{x}_{k+1}, \tilde{u}_{k+1}) + \cdots + g_{N-1}(\tilde{x}_{N-1}, \tilde{u}_{N-1}) + g_N(\tilde{x}_N).$$

Whereas the ordinary rollout algorithm would choose control $\tilde{u}_k$ and move to $\tilde{x}_{k+1}$, the fortified algorithm compares $C(\overline{T}_k)$ and $C(\tilde{T}_k)$, and depending on which of the two is smaller, chooses $\overline{u}_k$ or $\tilde{u}_k$ and moves to $\overline{x}_{k+1}$ or to $\tilde{x}_{k+1}$, respectively. In particular, if $C(\overline{T}_k) \leq C(\tilde{T}_k)$ the algorithm sets the next state and corresponding tentative trajectory to

$$x_{k+1} = \overline{x}_{k+1}, \qquad \overline{T}_{k+1} = \{\overline{x}_{k+1}, \overline{u}_{k+1}, \ldots, \overline{u}_{N-1}, \overline{x}_N\},$$

and if $C(\overline{T}_k) > C(\tilde{T}_k)$ it sets the next state and corresponding tentative trajectory to

$$x_{k+1} = \tilde{x}_{k+1}, \qquad \overline{T}_{k+1} = \{\tilde{x}_{k+1}, \tilde{u}_{k+1}, \ldots, \tilde{u}_{N-1}, \tilde{x}_N\}.$$

In other words the fortified rollout at x_k follows the current trajectory $\overline{T}_k$ unless a lower cost trajectory $\tilde{T}_k$ is discovered by running the base heuristic from all possible next states x_{k+1}. It follows that at every state the trajectory that consists of the union of the permanent and the tentative trajectories, has lower cost than the initial tentative trajectory, which is the one produced by the base heuristic starting from x_0. Moreover, it can be seen that if the base heuristic is sequentially improving, the rollout algorithm and its fortified version coincide.

Finally we note that the fortified rollout may be viewed as the ordinary rollout algorithm applied to a modified version of the original problem and modified base heuristic that has the sequential improvement property. The corresponding construction is somewhat tedious and will not be given; we refer to [BTW97] and [Ber17], Section 6.4.2.

Using Multiple Heuristics

In many problems, several promising heuristics may be available. It is then possible to use all of these heuristics in the rollout framework. The idea is to construct a *superheuristic*, which selects the best trajectory produced by all the base heuristic trajectories. The superheuristic can then be used as the base heuristic for a rollout algorithm.

In particular, let us assume that we have M base heuristics, and that the mth of these, given a state x_{k+1}, produces a trajectory

$$\tilde{T}^m_{k+1} = \{x_{k+1}, \tilde{u}^m_{k+1}, \ldots, \tilde{u}^m_{N-1}, \tilde{x}^m_N\},$$

and corresponding cost $C(\tilde{T}^m_{k+1})$. The superheuristic then produces at x_{k+1} the trajectory $\tilde{T}^m_{k+1}$ for which $C(\tilde{T}^m_{k+1})$ is minimum.

An interesting property, which can be readily verified by using the definitions, is that if all the base heuristics are sequentially improving, the same is true for the superheuristic. Moreover, there is a fortified version of the rollout algorithm, which has the property that it produces a trajectory with no worse cost than all the trajectories produced by the M base heuristics when started at the initial state x_0.

Rollout Algorithms with Multistep Lookahead

We may incorporate multistep lookahead into the deterministic rollout framework. To describe two-step lookahead, suppose that after k steps we have reached state x_k. We then consider the set of all two-step-ahead states x_{k+2} we run the base heuristic starting from each of them, and compute the two-stage cost to get from x_k to x_{k+2}, plus the cost of the base heuristic from x_{k+2}. We select the state, say $\tilde{x}_{k+2}$, that is associated with minimum cost, compute the controls $\tilde{u}_k$ and $\tilde{u}_{k+1}$ that lead from x_k to $\tilde{x}_{k+2}$, and choose $\tilde{u}_k$ as the next rollout control and $x_{k+1} = f_k(x_k, \tilde{u}_k)$ as the next state.

The extension of the algorithm to lookahead of more than two steps is straightforward: instead of the two-step-ahead states x_{k+2} we run the base heuristic starting from all the possible ℓ-step ahead states $x_{k+\ell}$, etc. A fortified version of the multistep rollout algorithm is possible along the lines described earlier. Moreover, for problems with a large number of stages, we can truncate the rollout trajectory and add a terminal cost function approximation to compensate for the resulting error.

Let us also note that in two-step lookahead rollout we may consider disregarding some of the two-step ahead states x_{k+2} that are judged less promising according to some criterion (for example the costs of the base heuristic from a one-step lookahead). Similarly in ℓ-step lookahead rollout it is possible to disregard some of the states that are ℓ steps or less ahead; see Fig. 2.4.3. This may be viewed as *selective depth lookahead*, and aims to limit the number of times that the base heuristic is applied, which can become overwhelming as the length of lookahead is increased. We will encounter again the idea of selective depth lookahead in the context of stochastic rollout and Monte Carlo tree search (see the next section), where in addition to the length of lookahead, the accuracy of the simulation to evaluate the cost of the base heuristic is adapted to the results of earlier computations.

2.4.2 Stochastic Rollout and Monte Carlo Tree Search

We will now discuss the extension of the rollout algorithm to stochastic problems. We will restrict ourselves to the case where the base heuristic is a policy $\pi = \{\mu_0, \ldots, \mu_{N-1}\}$ (i.e., is sequentially consistent, in the context of deterministic problems). It is possible to consider more general rollout algorithms that involve base heuristics with a sequential improvement property, but we will not pursue this idea, as it does not seem that it has been applied so far in interesting practical contexts.

We first note that the cost improvement property that we showed for deterministic problems under the sequential consistency condition carries through for stochastic problems. In particular, denote by $J_{k,\pi}(x_k)$ the cost corresponding to starting the base policy at state x_k, and by $J_{k,\tilde{\pi}}(x_k)$ the

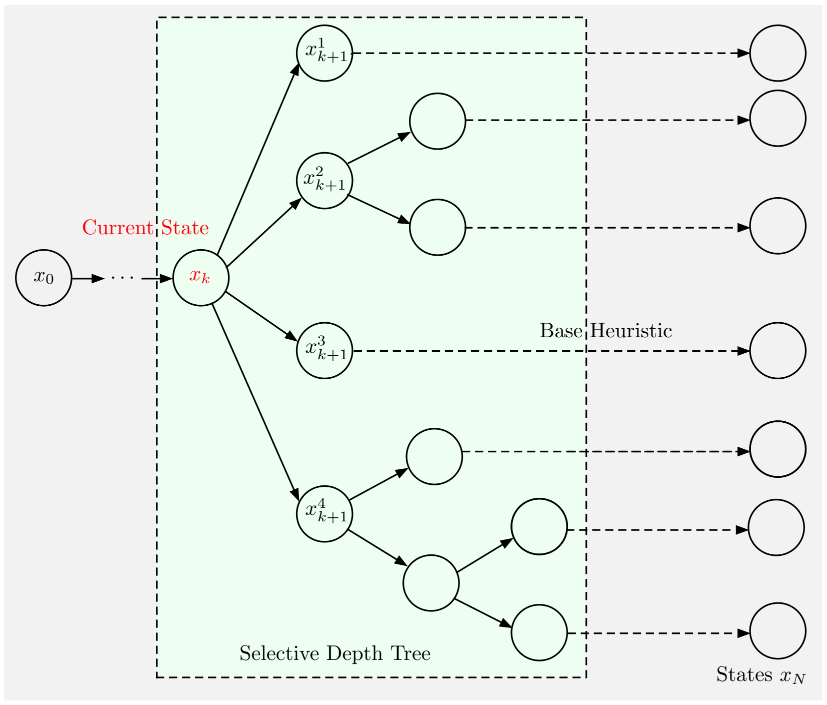

Figure 2.4.3 Illustration of deterministic rollout with selective depth lookahead. After k steps of the algorithm, we have a trajectory that states at the initial state x_0 and ends at state x_k. We then generate the set of all possible next states (states x_{k+1}^1, x_{k+1}^2, x_{k+1}^3, x_{k+1}^4 in the figure). We "evaluate" these states using the base heuristic, and select some of them for "expansion," i.e., we generate their next states x_{k+2}, evaluate them using the base heuristic, and continue. In the end we have a selective depth tree of next states, and the base heuristic costs from the leaves of the tree. The state x_{k+1} that corresponds to smallest overall cost is chosen by the selective depth lookahead rollout algorithm. For problems with a large number of stages, we can truncate the rollout trajectories and add a terminal cost function approximation as compensation for the resulting error.

cost corresponding to starting the rollout algorithm at state x_k. We claim that

$$J_{k,\tilde{\pi}}(x_k) \le J_{k,\pi}(x_k), \qquad \text{for all } x_k \text{ and } k.$$

We prove this inequality by induction similar to the deterministic case. Clearly it holds for $k = N$, since $J_{N,\tilde{\pi}} = J_{N,\pi} = g_N$. Assuming that it holds for index $k+1$, we have for all x_k,

$$\begin{aligned} J_{k,\tilde{\pi}}(x_k) &= E\Big\{g_k\big(x_k, \tilde{\mu}_k(x_k), w_k\big) + J_{k+1,\tilde{\pi}}\Big(f_k\big(x_k, \tilde{\mu}_k(x_k), w_k\big)\Big)\Big\} \\ &\le E\Big\{g_k\big(x_k, \tilde{\mu}_k(x_k), w_k\big) + J_{k+1,\pi}\big(f_k(x_k, \tilde{\mu}_k(x_k), w_k)\big)\Big\} \end{aligned}$$

$$
\begin{aligned}
&= \min_{u_k \in U_k(x_k)} E\Big\{g_k(x_k, u_k, w_k) + J_{k+1,\pi}\big(f_k(x_k, u_k, w_k)\big)\Big\} \\
&\leq E\Big\{g_k\big(x_k, \mu_k(x_k), w_k\big) + J_{k+1,\pi}\big(f_k(x_k, u_k, w_k)\big)\Big\} \\
&= J_{k,\pi}(x_k),
\end{aligned}
$$

where:

(a) The first equality is the DP equation for the rollout policy $\tilde{\pi}$.

(b) The first inequality holds by the induction hypothesis.

(c) The second equality holds by the definition of the rollout algorithm.

(d) The third equality is the DP equation for the policy π that corresponds to the base heuristic.

The induction proof of the cost improvement property (2.26) is thus complete.

Similar to deterministic problems, it has been observed empirically that for stochastic problems the rollout policy not only does not deteriorate the performance of the base policy, but also typically produces substantial cost improvement; see the case studies referenced at the end of the chapter.

Simulation-Based Implementation of the Rollout Algorithm

A conceptually straightforward way to compute the rollout control at a given state x_k and time k is to consider each possible control $u_k \in U_k(x_k)$ and to generate a "large" number of simulated trajectories of the system starting from (x_k, u_k). Thus a simulated trajectory has the form

$$x_{i+1} = f_i\big(x_i, \mu_i(x_i), w_i\big), \qquad i = k+1, \ldots, N-1,$$

where $\{\mu_{k+1}, \ldots, \mu_{N-1}\}$ is the tail portion of the base policy, the first generated state is

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

and the disturbance sequences $\{w_k, \ldots, w_{N-1}\}$ are independent random samples. The costs of the trajectories corresponding to a pair (x_k, u_k) can be viewed as samples of the Q-factor

$$Q_k(x_k, u_k) = E\Big\{g_k(x_k, u_k, w_k) + J_{k+1,\tilde{\pi}}\big(f_k(x_k, u_k, w_k)\big)\Big\},$$

where $J_{k+1,\tilde{\pi}}$ is the cost-to-go function of the base policy, i.e., $J_{k+1,\tilde{\pi}}(x_{k+1})$ is the cost of using the base policy starting from x_{k+1}. By Monte Carlo averaging of the costs of the sample trajectories corresponding to each control $u_k \in U_k(x_k)$, we obtain an approximation to $Q_k(x_k, u_k)$, which we denote by $\tilde{Q}_k(x_k, u_k)$. Then the (approximate) rollout control $\tilde{\mu}_k(x_k)$ is computed with the minimization

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

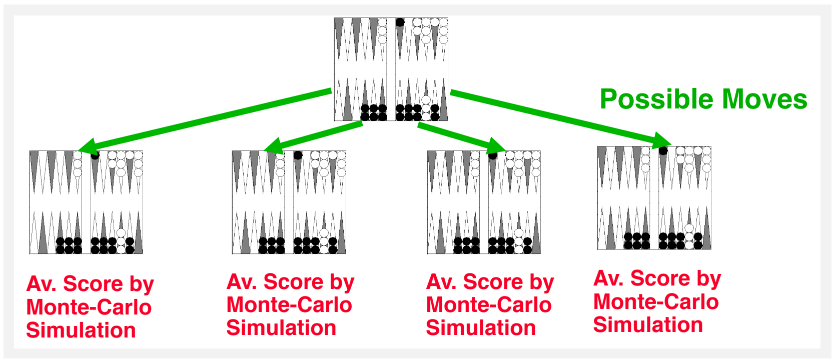
Possible Moves
Av. Score by Monte-Carlo Simulation
Av. Score by Monte-Carlo Simulation
Av. Score by Monte-Carlo Simulation
Av. Score by Monte-Carlo Simulation

Figure 2.4.4 Illustration of rollout for backgammon. At a given position and roll of the dice, the set of all possible moves is generated, and the outcome of the game for each move is evaluated by "rolling out" (simulating to the end) many games using a suboptimal backgammon player (the base heuristic), and by Monte-Carlo averaging the scores. The move that results in the best average score is selected for play.

Example 2.4.2 (Backgammon)

The first impressive application of rollout was in the game of backgammon by Tesauro and Galperin [TeG96], see Fig. 2.4.4. They implemented a rollout algorithm, which attained a level of play that was better than all computer backgammon programs, and eventually better than the best humans. Tesauro had proposed earlier the use of neural networks as cost function approximators for backgammon, resulting in a program called TD-Gammon [Tes89a], [Tes89b], [Tes92], [Tes94], [Tes95], [Tes02]. TD-Gammon was trained with the use of the TD(λ) algorithm that will be discussed in Section 4.9. The rollout algorithm also involved multi-step lookahead of a small number of stages, as well as truncation of long rollout trajectories, using a terminal cost function approximation based on TD-Gammon.

At present, most computer backgammon programs descend from TD-Gammon. Rollout-based backgammon programs are the most powerful in terms of performance, consistent with the principle that a rollout algorithm performs better than its base heuristic. However, rollout-based backgammon programs are too time-consuming for real-time play at present, because of the extensive on-line simulation requirement at each move. They have been used in a limited diagnostic way to assess the quality of neural network-based programs (a list of articles on computer backgammon may be found at http://www.bkgm.com/articles/page07.html).

In the rollout implementation just described, we implicitly assumed that once we reach state x_k, we generate the same large number of trajectories starting from each pair (x_k, u_k), with $u_k \in U(x_k)$, to the end of the horizon. The drawback of this is threefold:

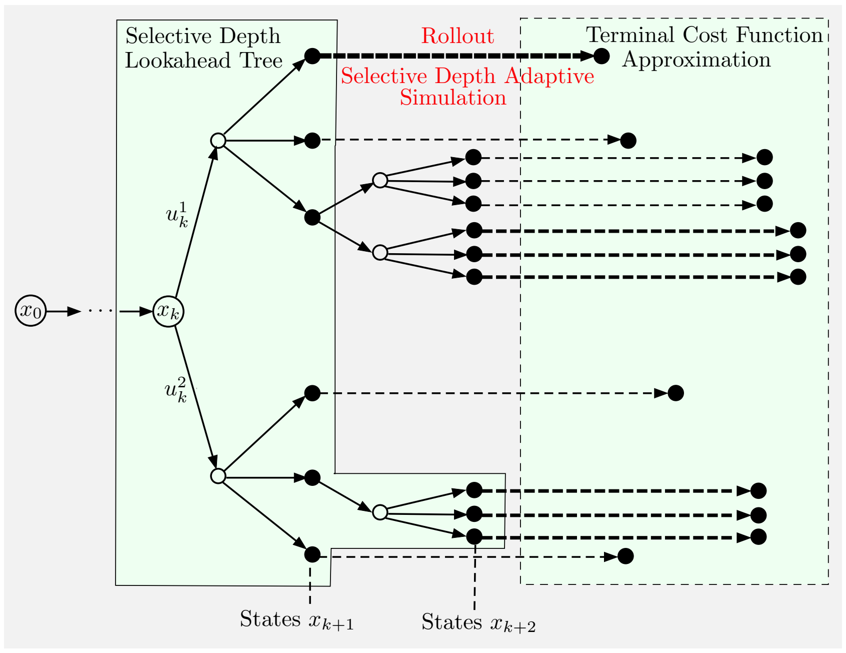
Selective Depth Lookahead Tree
Rollout
Selective Depth Adaptive Simulation
Terminal Cost Function Approximation
x_0
x_k
u_k^1
u_k^2
States x_{k+1}
States x_{k+2}

Figure 2.4.5 Illustration of a tree used by MCTS at a state x_k. We assume for simplicity two available controls u_k^1 and u_k^2, and three values of disturbance at each stage. For some of the possible states x_{k+1} we use an additional step lookahead and/or discard one of the two controls, thus generating the selective depth tree shown. We approximate the cost-to-go of the base policy from the leaf-states of the tree by generating a variable (perhaps adaptively determined) number of rollout trajectories, using the base policy, with possibly limited horizon, combined with terminal cost function approximation. Based on these results, we evaluate the approximate Q-factors corresponding to the two controls u_k^1 and u_k^2, and we select the one with smallest Q-factor.

(a) The trajectories may be too long because the horizon length N is large (or infinite, in an infinite horizon context).

(b) Some of the controls u_k may be clearly inferior to others, and may not be worth as much sampling effort.

(c) Some of the controls u_k that appear to be promising, may be worth exploring better through multistep lookahead.

This has motivated variants, generally referred to as *Monte Carlo tree search* (MCTS for short), which aim to trade off computational economy with a hopefully small risk of degradation in performance. Variants of this type involve, among others, early discarding of controls deemed to be inferior based on the results of preliminary calculations, and simulation that is limited in scope (either because of a reduced number of simulation samples, or because of a shortened horizon of simulation, or both); see Fig. 2.4.5.

In particular, a simple remedy for (a) above is to use rollout trajectories of reasonably limited length, perhaps with some terminal cost function at the end. This terminal cost function may be obtained through some auxiliary calculation, such as cost-to-go approximation at a given level of lookahead. In fact the base policy used for rollout may be the one that provides the terminal cost function approximation, as noted for the rollout-based backgammon algorithm of Example 2.4.2.

A simple but less straightforward remedy for (b) is to use some heuristic or statistical test to discard some of the controls u_k, as soon as this is suggested by the early results of simulation. Similarly, to implement (c) one may use some heuristic to increase the length of lookahead selectively for some of the controls u_k. This is similar to the selective depth lookahead procedure for deterministic rollout that we illustrated in Fig. 2.4.3.

The MCTS approach can be based on sophisticated procedures for implementing and combining the ideas just described. These procedures use among others the interim results of the computation and statistical (confidence interval) tests to focus the simulation effort along the most promising directions. Still, however, there is no general theory, and the implementation of MCTS is often ad hoc and problem dependent; we refer to the end-of-chapter literature.

A major success has been the use of MCTS in the AlphaGo and AlphaZero computer programs (Silver et al. [SHM16], [SHS17]), which perform better than the best humans in the games of Go and chess. These programs integrates several of the techniques discussed in this chapter, and in Chapters 3 and 4, including off-line parametric cost approximation techniques that use deep neural networks, rollout using an off-line trained base policy followed by cost function approximation, and associated MCTS variations. The base policy was trained using a deep neural network, using techniques that we will discuss in Chapter 3.

Variance Reduction in Rollout

When using simulation, sampling is often organized to effect *variance reduction*. By this we mean that for a given problem, the collection and use of samples is structured so that the variance of the simulation error is made smaller, with the same amount of simulation effort. There are several methods of this type for which we refer to textbooks on simulation (see, e.g., Ross [Ros12], Rubinstein and Kroese [RuK17]).

In this section we discuss a method to reduce the effects of the simulation error in the calculation of the Q-factors in the context of rollout. The key idea is that the selection of the rollout control depends on the values of the Q-factor differences

$$Q_k(x_k, u_k) - Q_k(x_k, \hat{u}_k)$$

for all pairs of controls $(u_k, \hat{u}_k)$. These values must be computed accurately, so that the controls u_k and $\hat{u}_k$ can be accurately compared. On the other hand, the simulation/approximation errors in the computation of the individual Q-factors $Q_k(x_k, u_k)$ may be magnified through the preceding differencing operation.

An alternative approach is possible in the case where the probability distribution of each disturbance w_k does not depend on x_k and u_k. In this case, we may approximate by simulation the Q-factor difference $Q_k(x_k, u_k) - Q_k(x_k, \hat{u}_k)$ by sampling the difference

$$C_k(x_k, u_k, \mathbf{w}_k) - C_k(x_k, \hat{u}_k, \mathbf{w}_k),$$

where $\mathbf{w}_k = (w_k, w_{k+1}, \ldots, w_{N-1})$ and

$$C_k(x_k, u_k, \mathbf{w}_k) = g_N(x_N) + g_k(x_k, u_k, w_k) + \sum_{i=k+1}^{N-1} g_i\big(x_i, \mu_i(x_i), w_i\big),$$

where $\{\mu_{k+1}, \ldots, \mu_{N-1}\}$ is the tail portion of the base policy.

This approximation may be far more accurate than the one obtained by differencing independent samples $C_k(x_k, u_k, \mathbf{w}_k)$ and $C_k(x_k, \hat{u}_k, \hat{\mathbf{w}}_k)$. Indeed, by introducing the zero mean sample errors

$$D_k(x_k, u_k, \mathbf{w}_k) = C_k(x_k, u_k, \mathbf{w}_k) - Q_k(x_k, u_k),$$

it can be seen that the variance of the error in estimating $Q_k(x_k, u_k) - Q_k(x_k, \hat{u}_k)$ with the former method will be smaller than with the latter method if and only if

$$E_{\mathbf{w}_k, \hat{\mathbf{w}}_k}\Big\{\big|D_k(x_k, u_k, \mathbf{w}_k) - D_k(x_k, \hat{u}_k, \hat{\mathbf{w}}_k)\big|^2\Big\}$$
$$> E_{\mathbf{w_k}}\left\{\big|D_k(x_k, u_k, \mathbf{w}_k) - D_k(x_k, \hat{u}_k, \mathbf{w}_k)\big|^2\right\},$$

or equivalently

$$E\big\{D_k(x_k, u_k, \mathbf{w}_k) D_k(x_k, \hat{u}_k, \mathbf{w}_k)\big\} > 0; \tag{2.29}$$

i.e., if and only if the correlation between the errors $D_k(x_k, u_k, \mathbf{w}_k)$ and $D_k(x_k, \hat{u}_k, \mathbf{w}_k)$ is positive. A little thought should convince the reader that this property is likely to hold in many types of problems. Roughly speaking, the relation (2.29) holds if changes in the value of u_k (at the first stage) have little effect on the value of the error $D_k(x_k, u_k, \mathbf{w}_k)$ relative to the effect induced by the randomness of $\mathbf{w}_k$. To see this, suppose that there exists a scalar $\gamma < 1$ such that, for all x_k, u_k, and $\hat{u}_k$, there holds

$$E\left\{\big|D_k(x_k, u_k, \mathbf{w}_k) - D_k(x_k, \hat{u}_k, \mathbf{w}_k)\big|^2\right\} \leq \gamma E\left\{\big|D_k(x_k, u_k, \mathbf{w}_k)\big|^2\right\}. \tag{2.30}$$

Then we have

$$\begin{aligned}D_k(x_k,u_k,\mathbf{w}_k)D_k(x_k,\hat{u}_k,\mathbf{w}_k)\\&= \big|D_k(x_k,u_k,\mathbf{w}_k)\big|^2\\&\quad + D_k(x_k,u_k,\mathbf{w}_k)\big(D_k(x_k,\hat{u}_k,\mathbf{w}_k)-D_k(x_k,u_k,\mathbf{w}_k)\big)\\&\geq \big|D_k(x_k,u_k,\mathbf{w}_k)\big|^2\\&\quad - \big|D_k(x_k,u_k,\mathbf{w}_k)\big|\cdot\big|D_k(x_k,\hat{u}_k,\mathbf{w}_k)-D_k(x_k,u_k,\mathbf{w}_k)\big|,\end{aligned}$$

from which we obtain

$$\begin{aligned}E\big\{D_k(x_k,&u_k,\mathbf{w}_k)D_k(x_k,\hat{u}_k,\mathbf{w}_k)\big\}\\&\geq E\left\{\big|D_k(x_k,u_k,\mathbf{w}_k)\big|^2\right\}\\&\quad - E\Big\{\big|D_k(x_k,u_k,\mathbf{w}_k)\big|\cdot\big|D_k(x_k,\hat{u}_k,\mathbf{w}_k)-D_k(x_k,u_k,\mathbf{w}_k)\big|\Big\}\\&\geq E\left\{\big|D_k(x_k,u_k,\mathbf{w}_k)\big|^2\right\}-\frac{1}{2}E\left\{\big|D_k(x_k,u_k,\mathbf{w}_k)\big|^2\right\}\\&\quad -\frac{1}{2}E\Big\{\big|D_k(x_k,\hat{u}_k,\mathbf{w}_k)-D_k(x_k,u_k,\mathbf{w}_k)\big|^2\Big\}\\&\geq \frac{1-\gamma}{2}E\left\{\big|D_k(x_k,u_k,\mathbf{w}_k)\big|^2\right\},\end{aligned}$$

where for the first inequality we use the generic relation $ab \geq a^2 - |a|\cdot|b-a|$ for two scalars a and b, for the second inequality we use the generic relation $|a|\cdot|b| \geq -\frac{1}{2}(a^2+b^2)$ for two scalars a and b, and for the third inequality we use Eq. (2.30).

Thus, under the assumption (2.30) and the assumption

$$E\left\{\big|D_k(x_k,u_k,\mathbf{w}_k)\big|^2\right\} > 0,$$

the condition (2.29) holds and guarantees that by averaging cost difference samples rather than differencing (independently obtained) averages of cost samples, the simulation error variance decreases.

2.4.3 Model Predictive Control

In many control problems where the objective is to keep the state of a system near some desired point, linear-quadratic models are not satisfactory. There are two main reasons for this:

(a) The system may be nonlinear, and using for control purposes a model that is linearized around the desired point may be inappropriate.

(b) There may be control and/or state constraints, which are not handled adequately through a quadratic penalty on state and control. In particular, there may be special structure of the problem dictating that, for efficiency purposes, the system should often be operated at the boundary of its constraints. The solution obtained from a linear-quadratic model is not suitable for this, because it does not impose directly constraints on the states and the controls and the quadratic penalty tends to "blur" the boundaries of the constraints.

These inadequacies of the linear-quadratic model have motivated a form of suboptimal control, called *model predictive control* (MPC), which combines elements of several ideas that we have discussed so far: certainty equivalence, multistep lookahead, and rollout algorithms. We will focus primarily on the most common form of MPC, where the system is either deterministic, or else it is stochastic, but it is replaced with a deterministic version by using typical values in place of all uncertain quantities, as in the certainty equivalent control approach. At each stage, a (deterministic) optimal control problem is solved over a fixed length horizon, starting from the current state. The first component of the corresponding optimal policy is then used as the control of the current stage, while the remaining components are discarded. The process is then repeated at the next stage, once the next state is revealed.

The primary objective in MPC, aside from fulfilling the state and control constraints of the problem, is to obtain a stable closed-loop system. Note here that we may only be able to guarantee the stability of the deterministic model that forms the basis for the calculations of the MPC. This is consistent with a common practice in control theory: design a stable controller for a deterministic model of the system, and expect that it will provide some form of stability in a realistic stochastic environment as well.

We will first discuss some issues of multistage lookahead, which are relevant to MPC, but are also important in a broader context. Then we will explain the mechanism by which stability is achieved by MPC under some reasonable conditions.

As noted earlier, model predictive control (MPC) was initially motivated by the desire to introduce nonlinearities, and control and/or state constraints into the linear-quadratic framework, and obtain a suboptimal but stable closed-loop system. With this in mind, we will describe MPC for the special case of a stationary, possibly nonlinear, deterministic system, where state and control belong to some Euclidean spaces. The methodology, however, can be extended to more general contexts (see the end-of-chapter references). The system is

$$x_{k+1} = f(x_k, u_k), \qquad k = 0, 1, \ldots,$$

and the cost per stage is quadratic:

$$x_k' Q x_k + u_k' R u_k, \qquad k = 0, 1, \ldots,$$

where Q and R are positive definite symmetric matrices. We impose state and control constraints

$$x_k \in X, \qquad u_k \in U(x_k), \qquad k = 0, 1, \ldots,$$

and we assume that the set X contains the origin of the corresponding Euclidean space. Furthermore, if the system is at the origin, it can be kept there at no cost with control equal to 0, i.e., $0 \in U(0)$ and $f(0,0) = 0$. We want to derive a stationary feedback controller that applies control $\tilde{\mu}(x)$ at state x, and is such that, for all initial states $x_0 \in X$, the state of the closed-loop system

$$x_{k+1} = f\big(x_k, \tilde{\mu}(x_k)\big),$$

satisfies the state and control constraints, and the total cost over an infinite number of stages is finite:

$$\sum_{k=0}^{\infty} \big(x_k' Q x_k + \tilde{\mu}(x_k)' R\, \tilde{\mu}(x_k)\big) < \infty. \tag{2.31}$$

This finite cost condition guarantees that tthe feedback controller $\tilde{\mu}$ is stable in the sense that $x_k \to 0$ and $\tilde{\mu}(x_k) \to 0$ for all initial states $x_0 \in X$.

In order for such a controller to exist, it is evidently sufficient [in view of the assumption that $f(0,0) = 0$] that *there exists a positive integer m such that for every initial state $x_0 \in X$, we can find a sequence of controls u_k, $k = 0, \ldots, m-1$, which drive to 0 the state x_m of the system at time m, while keeping all the preceding states $x_1, x_2, \ldots, x_{m-1}$ within X and satisfying the control constraints $u_0 \in U(x_0), \ldots, u_{m-1} \in U(x_{m-1})$.* We refer to this as the *constrained controllability assumption*. In practical applications, this assumption can often be checked easily.

Let us now describe a form of MPC under the preceding assumption. At each stage k and state $x_k \in X$, it solves an m-stage deterministic optimal control problem involving the same quadratic cost *and the requirement that the state after m stages is exactly equal to 0.* This is the problem

$$\min_{u_i,\, i=k,\ldots,k+m-1} \sum_{i=k}^{k+m-1} (x_i' Q x_i + u_i' R u_i), \tag{2.32}$$

subject to the system equation constraints

$$x_{i+1} = f(x_i, u_i), \qquad i = k, k+1, \ldots, k+m-1,$$

the state and control constraints

$$x_i \in X, \qquad u_i \in U(x_i), \qquad i = k, k+1, \ldots, k+m-1,$$

and the terminal state constraint

$$x_{k+m} = 0.$$

By the constrained controllability assumption, this problem has a feasible solution. Let $\{\tilde{u}_k, \tilde{u}_{k+1}, \ldots, \tilde{u}_{k+m-1}\}$ be a corresponding optimal control sequence. The MPC applies at stage k the first component of this sequence,

$$\tilde{\mu}(x_k) = \tilde{u}_k,$$

and discards the remaining components.

We can make the connection of MPC with rollout by noting that *the one-step lookahead function $\tilde{J}$ implicitly used by MPC [cf. Eq. (2.32)] is the cost-to-go function of a certain base policy.* This is the policy that drives to 0 the state after $m-1$ stages and keeps the state at 0 thereafter, while observing the state and control constraints, and minimizing the quadratic cost. Thus, we can also think of MPC as a rollout algorithm with the policy just described viewed as the base heuristic.

Stability Analysis

We now show that the MPC satisfies the stability condition (2.31). Let $x_0, u_0, x_1, u_1, \ldots$ be the state and control sequence generated by MPC, so that

$$u_k = \tilde{\mu}(x_k), \qquad x_{k+1} = f\big(x_k, \tilde{\mu}(x_k)\big), \qquad k = 0, 1, \ldots$$

Denote by $\hat{J}(x)$ the optimal cost of the m-stage problem solved by MPC when at a state $x \in X$. Let also $\tilde{J}(x)$ be the optimal cost starting at x of a corresponding $(m-1)$-stage problem, i.e., the optimal value of the quadratic cost

$$\sum_{k=0}^{m-2} (x_k' Q x_k + u_k' R u_k),$$

where $x_0 = x$, subject to the constraints

$$x_k \in X, \qquad u_k \in U(x_k), \qquad k = 0, \ldots, m-2,$$

and

$$x_{m-1} = 0.$$

[For states $x \in X$ for which this problem does not have a feasible solution, we write $\tilde{J}(x) = \infty$.] Since having one less stage in our disposal to drive the state to 0 cannot decrease the optimal cost, we have for all $x \in X$

$$\hat{J}(x) \leq \tilde{J}(x). \tag{2.33}$$

From the definitions of $\hat{J}$ and $\tilde{J}$, we have for all k,

$$\min_{u\in U(x)} \Big[x_k'Qx_k+u'Ru+\tilde{J}\big(f(x_k,u)\big)\Big] = x_k'Qx_k+u_k'Ru_k+\tilde{J}(x_{k+1}) = \hat{J}(x_k),$$

so using Eq. (2.33), we obtain

$$x_k'Qx_k + u_k'Ru_k + \hat{J}(x_{k+1}) \le \hat{J}(x_k), \qquad k = 0,1,\ldots.$$

Adding this equation for all k in a range $[0,K]$, where $K = 0,1,\ldots$, we obtain

$$\hat{J}(x_{K+1}) + \sum_{k=0}^{K}(x_k'Qx_k + u_k'Ru_k) \le \hat{J}(x_0).$$

Since $\hat{J}(x_{K+1}) \ge 0$, it follows that

$$\sum_{k=0}^{K}(x_k'Qx_k + u_k'Ru_k) \le \hat{J}(x_0), \qquad K = 0,1,\ldots, \tag{2.34}$$

and taking the limit as $K \to \infty$,

$$\sum_{k=0}^{\infty}(x_k'Qx_k + u_k'Ru_k) \le \hat{J}(x_0) < \infty.$$

This shows the stability condition (2.31).

Example 2.4.3

Consider the scalar linear system

$$x_{k+1} = x_k + u_k,$$

and the state and control constraints

$$x_k \in X = \big\{x \mid |x| \le 1.5\big\}, \qquad u_k \in U(x_k) = \big\{u \mid |u| \le 1\big\}.$$

Let also $Q = R = 1$. We select $m = 2$. For this value of m, the constrained controllability assumption is satisfied.

When at state $x_k \in X$, the MPC minimizes the two-stage cost

$$x_k^2 + u_k^2 + (x_k + u_k)^2 + u_{k+1}^2,$$

subject to the control constraints

$$|u_k| \le 1, \qquad |u_{k+1}| \le 1,$$

and the state constraints

$$x_{k+1} \in X, \qquad x_{k+2} = x_k + u_k + u_{k+1} = 0.$$

It is easily verified that this minimization yields $u_{k+1} = -(x_k + u_k)$ and $u_k = -(2/3)x_k$. Thus the MPC takes the form

$$\overline{\mu}(x_k) = -\frac{2}{3}x_k,$$

and the closed-loop system is

$$x_{k+1} = \frac{1}{3}x_k, \qquad k = 0, 1, \ldots$$

Note that while the closed-loop system with this MPC is stable, its state is never driven to 0 if started from $x_0 \neq 0$.

Regarding the choice of the horizon length m for the MPC calculations, note that if the constrained controllability assumption is satisfied for some value of m, it is also satisfied for all larger values of m. Furthermore, it can be seen that the m-stage cost $\hat{J}(x)$, which by Eq. (2.34), is an upper bound to the cost of MPC, cannot increase with m. This argues for a larger value of m. On the other hand, the optimal control problem solved at each stage by the MPC becomes larger and hence more difficult as m increases. Thus, the horizon length is usually chosen on the basis of some experimentation: first ensure that m is large enough for the constrained controllability assumption to hold, and then by further experimentation to ensure overall satisfactory performance.

The MPC scheme that we have described is just the starting point for a broad methodology with many variations, which often relate to the suboptimal control methods that we have discussed so far in this chapter. For example, in the problem solved by MPC at each stage, instead of the requirement of driving the system state to 0 in m steps, one may use a large penalty for the state being nonzero after m steps. Then, the preceding analysis goes through, as long as the terminal penalty is chosen so that Eq. (2.33) is satisfied. In another variant one may use a nonquadratic cost function, which is everywhere positive except at $(x, u) = (0, 0)$. In still another variant, instead of aiming to drive the state to 0 after m steps, one aims to reach a sufficiently small neighborhood of the origin, within which a stabilizing controller, designed by other methods, may be used. Finally, we may consider variants of MPC methods, essentially combinations of problem approximation, certainty equivalent control, and rollout, which can deal with uncertainty and system disturbances; see the end-of-chapter references.

2.5 NOTES, SOURCES, AND EXERCISES

Approximation in value space has been considered in an ad hoc manner since the early days of DP and digital computation, motivated by the curse of dimensionality. The idea was reframed and coupled with model-free simulation methods that originated in the late 1980s in artificial intelligence. Since that time, approximation in value space has been one of the two pillars of approximate DP/RL. The other pillar is approximation in policy space, which will be receiving less attention in this book.

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 2.4.2. 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.

The application of rollout algorithms to discrete deterministic optimization problems has its origin in the neuro-dynamic programming work of the author and J. Tsitsiklis [BeT96], and has been further formalized by Bertsekas, Tsitsiklis, and Wu [BTW97], Bertsekas [Ber97], and Bertsekas and Castanon [BeC99]. A discussion of rollout algorithms as applied to network optimization problems may be found in the author's textbook [Ber98].

For more recent work on 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], and Lan, Guan, and Wu [LGW16]. These works discuss a broad variety of applications and case studies, and generally report positive computational experience. For a recent survey by the author, see [Ber13b].

The idea of rollout that uses limited lookahead, adaptive pruning of the lookahead tree, and cost function approximation at the end of the lookahead horizon 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 monograph by Chang, Hu, Fu, and Marcus [CFH05], [CFH13] proposed and analyzed adaptive sampling in connection with DP, including statistical tests to control the sampling process. The name "Monte Carlo tree search" (Section 6.5.1) has become popular, and in its current use, it encompasses adaptive sampling, rollout, extensions to sequential games, and the use and analysis of various statistical tests. We refer to the papers by Coulom [Cou06], and Kocsis and Szepesvari [KoS06], the surveys by Browne et al. [BPW12], and Runarsson, Schoenauer, and Sebag [RSS12], the monograph by Munos [Mun14], and the discussion by Fu [Fu17]. The technique for variance reduction in the calculation of Q-factor differences (Section 6.5.2) was given in the author's paper [Ber97].

The model predictive control approach is popular in a variety of control system design contexts, and particularly in chemical process control, where meeting explicit control and state constraints is an important practical issue. The connection of MPC with rollout algorithms was made in the author's paper [Ber05a]. The stability analysis given here is based on the work of Keerthi and Gilbert [KeG88]. For an early survey of the field, 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].

In our account of model predictive control, 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 [May14] for a review of various approaches that have been used in this context. The author's textbook [Ber17], Section 6.4, describes model predictive control for problems with set membership uncertainty and state constraints, using the concept of a target tube.